Festschrift

LNCS 5600

Alexander T. Borgida
Vinay K. Chaudhri
Paolo Giorgini
Eric S. Yu (Eds.)

Conceptual Modeling: Foundations and Applications

Essays in Honor of John Mylopoulos

Springer

Alexander T. Borgida   Vinay K. Chaudhri
Paolo Giorgini   Eric S. Yu (Eds.)

Conceptual Modeling: Foundations and Applications

Essays in Honor of John Mylopoulos

Springer

Volume Editors

Alexander T. Borgida
Rutgers University, Department of Computer Science
Piscataway, NJ 08855, USA
E-mail: borgida@cs.rutgers.edu

Vinay K. Chaudhri
SRI International, Artificial Intelligence Center
333 Ravenswood Ave, Menlo Park, CA, 94025, USA
E-mail: Vinay.Chaudhri@sri.com

Paolo Giorgini
University of Trento, Dipartimento Ingegneria e Scienza dell'Informazione
Via Sommarive, 14, 38100 Trento, Italy
E-mail: paolo.giorgini@unitn.it

Eric S. Yu
University of Toronto, Faculty of Information
140 St. George Street, Toronto, Ontario, M5S 3G6, Canada
E-mail: eric.yu@utoronto.ca

The cover illustration depicts the owl of Athena on a tetradrachm of Athens.
Permission to reproduce this image has been obtained from Krause Publications.

Library of Congress Control Number: Applied for

CR Subject Classification (1998): H.3.5, H.5.3, D.2, D.3, I.2.11, I.2.4

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

ISSN 0302-9743
ISBN-10 3-642-02462-9 Springer Berlin Heidelberg New York
ISBN-13 978-3-642-02462-7 Springer Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 12697083 06/3180 5 4 3 2 1 0

John Mylopoulos

Preface

John Mylopoulos has made ground-breaking contributions to three areas of computer science: artificial intelligence, information systems and software engineering.

His contributions have been celebrated on multiple occasions. First, Misha Missikoff organized a one-day symposium on conceptual modeling on June 17, 2003, in Velden, Austria, to celebrate John's 60th birthday. Second, John Tsotsos led the organization of a day of celebrations on June 27th, 2009 in Toronto, Canada, on the occasion of John's retirement from the Department of Computer Science of the University of Toronto.

This book grew out of our desire to honor and thank John by presenting him at the Toronto reunion with a volume that reflects his belief that conceptual modeling is becoming a fundamental skill that will be a necessary tool for all future computer scientists. The papers in this book are written by leading figures in technical areas that intersect with conceptual modeling, as well as by John's closest collaborators. We are pleased to present this collection of papers that we believe are of lasting significance and could also be used to support a course on conceptual modeling. We are extremely grateful to the eminent authors, who have contributed such high-quality material.

We have organized the chapters into several sections. Within each section, the chapters are ordered alphabetically by the surname of the first author. The section on foundations contains material on ontologies and knowledge representation, which we see as the technical grounding on which CM research builds – a pattern that John Mylopoulos himself has repeatedly followed, starting from semantic networks in the 1970's, through Reiter's solution to the frame problem, to the recent work of McIlraith on preferences in planning. The four sections on software and requirements engineering, information systems, information integration, and web and services, represent the chief current application domains for conceptual modeling[1]. Finally, the section on implementations discusses projects that build tools to support conceptual modeling. We note that the above divisions are by no means perfect, and several chapters could easily have been placed in more than one section.

Once again, we wish to express our gratitude to the authors, who have found time in their busy schedules to write these valuable chapters. We also wish to thank the referees, both authors and non-authors, who offered useful comments towards improving the material. We are grateful to the members of our Senior Advisory Committee, composed of Norm Badler, Sol Greenspan, Hector

[1] We point to the article by Roussopoulos and Karagianis for a history of some of the high points of the field.

Levesque, Nick Roussopoulos, John Tsotsos and Matthias Jarke, who provided us with sage advice and useful guidance at several crucial stages of this endeavor.

April 2009

Alex Borgida
Vinay Chaudhri
Paolo Giorgini
Eric Yu

Enkomion[2]

John Mylopoulos was born in Greece in 1943, and went to the United States to complete his B.Eng and M.Eng in Electrical Engineering at Brown University, and then do his PhD studies at Princeton University, under the direction of Theodosius Pavlidis, finishing in 1970.

John then joined the Department of Computer Science at the University of Toronto as an Assistant Professor, and he remained on the faculty there until his retirement in June 2009. In 2001, John joined as a visiting professor, and then in 2005 as Distinguished Professor, the University of Trento, in Trento, Italy.

Throughout his career John has had the unwavering support of his wife, Chryss, and they have raised two wonderful children, Maria and Myrto, who have followed their parents' footsteps into academia.

John is widely recognized as a visionary thinker. His insights, which cover the breadth of computer science, are much sought after. His keynotes are prescient and much-anticipated for a glimpse at the next big idea.

With great generosity, John has helped numerous young researchers get established, mentoring students and postdoctoral fellows. He has helped new departments and research groups gain prominence on the world stage.

He is a builder of communities and has worked tirelessly to bring people from diverse areas together in joint projects, creating much-needed synergy.

On a personal level, he is a role model: he is approachable, gentle, amiable and an eternal optimist, in other words, exactly the kind of person one would like to work with. His productivity is legendary. His leadership style is low-key but extremely effective.

Rather than provide a complete bibliography of all his publications and honors (which can be found online), or list all the many scientific contributions of John, we have chosen to offer three representative glimpses of his *opus*.

First, the following is a list of some of the projects that John undertook with his students and collaborators, and which were deemed sufficiently worthy for a Greek name (always starting with a "T") – John is very proud of his heritage:

– TORUS: Natural-language access to databases, which required the representation of the semantics of the data, and hence first led us to conceptual models of relational tables using semantic networks.

– TAXIS: Programming language for data-intensive applications which supported classes of objects, transactions, constraints, exceptions and workflows, all orthogonally organized in sub-class hierarchies with property inheritance.

[2] This is the original Greek source of the English word "encomium", meaning "celebration".

– TELOS: Representation language for knowledge of many different kinds of software engineering stakeholders, including application domain and development domain, which exploited meta-classes, and treated properties as objects.
– TROPOS: Applying the ideas of early requirements (goal orientation, agent dependence) to the entire range of software development, and expanding its scope to many topics, including security and evolution.

Second, in order to show that John not only worked in multiple areas but was in fact recognized in each of them as a leading figure, we mention three honors:

– Artificial Intelligence: Fellow of the American Association for Artificial Intelligence (1993).
– Databases: Elected President of the Very Large Databases Endowment (1998-01,2002-05), which oversees the VLDB conference and journal.
– Software and Requirements Engineering: Co-Editor-in-Chief of the Requirements Engineering Journal (2000-2008).

In addition, John was elected in 2007 a Fellow of the Royal Society of Canada, Academy of Sciences, Division of Applied Sciences and Engineering.

Finally, we provide the following genealogic tree of John's PhD students, and their "descendants", as of April 2009, as a testimony to both his skills as an advisor and in selecting some outstandig PhD students.

April 2009

Alex Borgida
Vinay Chaudhri
Paolo Giorgini
John Tsotsos
Eric Yu

The Academic Tree of John Mylopoulos

- John P. Mylopoulos
  Ph.D. 1970, Princeton University
  - Tourlakis George
    Ph.D. 1973, University of Toronto
  - Norman I. Badler
    Ph.D. 1975, University of Toronto
    - Bulent Ozguc
      Ph.D. 1978, University of Pennsylvania
      - Ugur Gudakbay
        Ph.D. 1994, Bilkent University
        - Mehmet Emin Donderler
          Ph.D. 2002, Bilkent University
        - Turker Yilmaz
          Ph.D. 2007, Bilkent University
      - Veysi Isler
        Ph.D. 1995, Bilkent University
      - Leyla Ozcivelek Durlu
        Ph.D. 1996, Bilkent University
      - Burcu Senyapili
        Ph.D. 1998, Bilkent University
      - Benal Tanrisever
        Ph.D. 2001, Bilkent University
      - Dilek Kaya Mutlu
        Ph.D. 2002, Bilkent University
      - Kaya Ozkaracalar
        Ph.D. 2004, Bilkent University
    - Larry Grim
      Ph.D. 1980, University of Pennsylvania
    - Joseph O'Rourke
      Ph.D. 1980, University of Pennsylvania
      - Alok Aggarwal
        Ph.D. 1984, Johns Hopkins University
      - Adlai DePano
        Ph.D. 1987, Johns Hopkins University
      - Subhash Suri
        Ph.D. 1987, Johns Hopkins University
        - Yunhong Zhou
          Ph.D. 2000, Washington University, St. Loius
        - Anshul Kothari
          Ph.D. 2005, University of California, Santa Barbara
        - Nisheeth Shrivastava
          Ph.D. 2006, University of California, Santa Barbara
        - Chiranjeeb Buragohain
          Ph.D. 2006, University of California, Santa Barbara

- Mylopoulos
  - Badler
    - O'Rourke
      - Michael McKenna
        Ph.D. 1988, Johns Hopkins University
      - Yan Ke
        Ph.D. 1989, Johns Hopkins University
      - Catherine Schevon
        Ph.D. 1989, Johns Hopkins University
      - Harold Conn
        Ph.D. 1990, Johns Hopkins University
      - Matthew Diaz
        Ph.D. 1991, Johns Hopkins University
    - Chaim Broit
      Ph.D. 1981, University of Pennsylvania
    - Sakunthala Gnanamgari
      Ph.D. 1981, University of Pennsylvania
    - Dan Olsen
      Ph.D. 1981, University of Pennsylvania
      - Olsen
        - Thomas McNeil
          Ph.D. 1993, Brigham Young University
        - Douglas Kohlert
          Ph.D. 1995, Brigham Young University
        - Ken Rodham
          Ph.D. 1995, Brigham Young University
    - James Korein
      Ph.D. 1984, University of Pennsylvania
    - Gerald Radack
      Ph.D. 1984, University of Pennsylvania
    - Stephen Platt
      Ph.D. 1985, University of Pennsylvania
    - Paul Fishwick
      Ph.D. 1986, University of Pennsylvania
      - Fishwick
        - Steven Walczak
          Ph.D. 1990, University of Florida
        - Victor Todd Miller
          Ph.D. 1993, University of Florida
        - Gyooseok Kim
          Ph.D. 1998, University of Florida
        - Kangsun Lee
          Ph.D. 1998, University of Florida
        - Robert Cubert
          Ph.D. 1999, University of Florida
        - Taewoo Kim
          Ph.D. 2002, University of Florida
        - Jin Joo Lee
          Ph.D. 2005, University of Florida

- Mylopoulos
  - Badler
    - Fishwick
      - Minho Park
        Ph.D. 2005, University of Florida
      - Hyunju Shim
        Ph.D. 2006, University of Florida
    - Tamar Granor
      Ph.D. 1986, University of Pennsylvania
    - Pearl Pu
      Ph.D. 1989, University of Pennsylvania
    - Diana Dadamo
      Ph.D. 1990, University of Pennsylvania
    - Jugal Kalita
      Ph.D. 1990, University of Pennsylvania
    - Kalita
      - Gerardo Perez Gonzalez
        Ph.D. 2003, University of Colorado, Colorado Springs
      - Lori De Looze
        Ph.D. 2005, University of Colorado, Colorado Springs
      - Utpal Sharma
        Ph.D. 2007, University of Colorado, Colorado Springs
      - Mona Soliman Habib
        Ph.D. 2008, University of Colorado, Colorado Springs
    - Isaac Rudomin
      Ph.D. 1990, University of Pennsylvania
    - Rudomin
      - Marissa Diaz Pier
        Ph.D. 2007, Instituto Technologico y de Estudios Superiores de Monterrey
      - Erik Millan
        Ph.D. 2008, Instituto Technologico y de Estudios Superiores de Monterrey
    - Susanna Wei
      Ph.D. 1990, University of Pennsylvania
    - Catherine Pelachaud
      Ph.D. 1991, University of Pennsylvania
    - Pelachaud
      - Radoslaw Niewiadomski
        Ph.D. 2007, University of Perugia
      - Magalie Ochs
        Ph.D. 2007, University of Paris 8
      - Maurizio Mancini
        Ph.D. 2008, Univeristy of Rome 3
    - Tarek Alameldin
      Ph.D. 1991, University of Pennsylvania
    - Cary Philips
      Ph.D. 1991, University of Pennsylvania
    - Moon Jung
      Ph.D. 1992, University of Pennsylvania
    - Wallace Ching
      Ph.D. 1992, University of Pennsylvania

- Mylopoulos
  - Badler
    - Eunyoung Koh
      Ph.D. 1993, University of Pennsylvania
    - Philip Lee
      Ph.D. 1993, University of Pennsylvania
    - Jianmin Zhao
      Ph.D. 1993, University of Pennsylvania
    - Hyeongseok Ko
      Ph.D. 1994, University of Pennsylvania
      - Ko
        - Kwang-Jin Choi
          Ph.D. 2003, Seoul National University
        - Byoungwon Choe
          Ph.D. 2004, Seoul National University
        - Oh-young Song
          Ph.D. 2004, Seoul National University
        - Seyoon Tak
          Ph.D. 2004, Seoul National University
    - Welton Becket
      Ph.D. 1996, University of Pennsylvania
    - Francisco Azuola
      Ph.D. 1996, University of Pennsylvania
    - Paul Diefenbach
      Ph.D. 1996, University of Pennsylvania
    - Libby Levison
      Ph.D. 1996, University of Pennsylvania
    - Min-Zhi Shao
      Ph.D. 1996, University of Pennsylvania
    - Xinmin Zhao
      Ph.D. 1996, University of Pennsylvania
    - Barry Reich
      Ph.D. 1997, University of Pennsylvania
    - Pei-Hwa Ho
      Ph.D. 1998, University of Pennsylvania
    - Bond-Jay Ting
      Ph.D. 1998, University of Pennsylvania
    - Deepak Tolani
      Ph.D. 1998, University of Pennsylvania
    - Diane Chi
      Ph.D. 1999, University of Pennsylvania
    - Sonu Chopra
      Ph.D. 1999, University of Pennsylvania
    - Rama Bindiganavale
      Ph.D. 2000, University of Pennsylvania
    - John Granieri
      Ph.D. 2000, University of Pennsylvania

Mylopoulos

- Badler
  - Jianping Shi
    Ph.D. 2000, University of Pennsylvania
  - Liwei Zhao
    Ph.D. 2001, University of Pennsylvania
  - Charles Erignac
    Ph.D. 2001, University of Pennsylvania
  - Rebecca Mercuri
    Ph.D. 2001, University of Pennsylvania
  - Suejung Huh
    Ph.D. 2002, University of Pennsylvania
  - Sooha Lee
    Ph.D. 2002, University of Pennsylvania
  - Koji Ashida
    Ph.D. 2003, University of Pennsylvania
  - Ying Liu
    Ph.D. 2003, University of Pennsylvania
  - Aaron Bloomfield
    Ph.D. 2004, University of Pennsylvania
  - Erdan Gu
    Ph.D. 2006, University of Pennsylvania
  - Seung-Joo Lee
    Ph.D. 2006, University of Pennsylvania
  - Nuria Pelechano
    Ph.D. 2006, University of Pennsylvania
  - Michael Johns
    Ph.D. 2007, University of Pennsylvania
  - Durell Bouchard
    Ph.D. 2008, University of Pennsylvania
  - Jan Allbeck
    Ph.D. 2009, University of Pennsylvania
  - Liming Zhao
    Ph.D. 2009, University of Pennsylvania
- Nicholas Roussopoulos
  Ph.D. 1976, University of Toronto
- Roussopoulos
  - Jim Coolahn
    Ph.D. 1984, University of Maryland
  - Leo Mark
    Ph.D. 1985, Aarhus University
  - Hyunchul Kang
    Ph.D. 1987, University of Maryland
  - Kang
    - Chanho Moon
      Ph.D. 2003, Chung-Ang University
    - Youngsung Kim
      Ph.D. 2005, Chung-Ang University
    - Daehyun Hwang
      Ph.D. 2007, Chung-Ang University

- Mylopoulos
  - Roussopoulos
    - Alex Delis
      Ph.D. 1993, University of Maryland
      - Delis
        - Vinay Kanitkar
          Ph.D. 2000, Brooklyn Polytechnic University
        - George Kollios
          Ph.D. 2000, Brooklyn Polytechnic University
        - Je-Ho Park
          Ph.D. 2001, brooklyn Polytechnic University
        - Richard Regan
          Ph.D. 2001, Brooklyn Polytechnic University
        - Zhongqiang Chen
          Ph.D. 2001, Brooklyn Polytechnic University
        - Vassil Kriakov
          Ph.D. 2008, Brooklyn Polytechnic University
    - Chung-Min Chen
      Ph.D. 1994, University of Maryland
    - Kostas Stathatos
      Ph.D. 1998, University of Maryland
    - Yannis Kotidis
      Ph.D. 2000, University of Maryland
    - Manuel Rodriguez
      Ph.D. 2001, University of Maryland
    - Alexandros Labrinidis
      Ph.D. 2002, University of Maryland
      - Labrinidis
        - Mohamed Sharaf
          Ph.D. 2007, University of Pittsburgh
        - Huiming Qu
          Ph.D. 2007, University of Pittsburgh
    - Zhexuan (Jeff) Song
      Ph.D. 2003, University of Maryland
    - Yannis Sismanis
      Ph.D. 2004, University of Maryland
    - Antonis Deligiannakis
      Ph.D. 2005, University of Maryland
    - Dimitris Tsoumakos
      Ph.D. 2006, University of Maryland
  - Richard David Peacocke
    Ph.D. 1978, University of Toronto
  - Michael Bauer
    Ph.D. 1978, University of Toronto
    - Bauer
      - Jinhui Qin
        Ph.D. 2008, University of Western Ontario
      - Elvis Viera
        Ph.D. 2007, University of Western Ontario

- Mylopoulos
  - Bauer
    - Keith Edwards
      Ph.D. 2004, University of Western Ontario
    - Mechelle Gittens
      Ph.D. 2003, University of Western Ontario
    - Michael Katchabaw
      Ph.D. 2002, University of Western Ontario
    - Andrew Marshall
      Ph.D. 2000, University of Western Ontario
    - Hasina Abdu
      Ph.D. 2000, University of Western Ontario
    - Stephen Howard
      Ph.D. 1999, University of Western Ontario
    - Douglas Williams
      Ph.D. 1999, University of Western Ontario
    - Crispin Cowan
      Ph.D. 1995, University of Western Ontario
  - Douglas Skuce
    Ph.D. 1977, McGill University
    - Skuce
      - Timothy Christian Lethbridge
        Ph.D. 1994, University of Ottawa
        - Lethbridge
          - Iyad Zayour
            Ph.D. 2002, University of Ottawa
          - Jelber Sayyad
            Ph.D. 2003, University of Ottawa
          - Abdelwahab Hamout-Lhadj
            Ph.D. 2005, University of Ottawa
          - Adam Murray
            Ph.D. 2006, University of Ottawa
      - Branka Tauzovich
        Ph.D. 1988, University of Ottawa
  - John K. Tsotsos
    Ph.D. 1980, University of Toronto
    - Tsotos
      - Testsutaro Shibahara
        Ph.D. 1985, University of Toronto
      - Ron Gershon
        Ph.D. 1987, University of Toronto
      - John Barron
        Ph.D. 1988, University of Toronto
        - Barron
          - Steven Beauchemin
            Ph.D. 1997, University of Western Ontario
          - Baozhong Tian
            Ph.D. 2006, University of Western Ontario

- Mylopoulos
  - Tsotos
    - Michael R.M. Jenkin
      Ph.D. 1988, University of Toronto
      - Jenkin
        - Bill Kapralos
          Ph.D. 2006, York University
        - Daniel C. Zikovitz
          Ph.D. 2004, York University
        - Iraj Mantegh
          Ph.D. 1998, University of Toronto
        - Andrew Hogue
          Ph.D. 2008, York University
    - Gregory Dudek
      Ph.D. 1991, University of Toronto
      - Dudek
        - Ioannis Rekleitis
          Ph.D. 2003, McGill University
        - Robert Sim
          Ph.D. 2004, McGill University
        - Saul (Shlomo) SImhon
          Ph.D. 2006, McGill University
        - Lub-Abril Torres-Mendez
          Ph.D. 2005, McGill University
        - Eric Bourque
          Ph.D. 2005, McGill University
        - Dimitris Marinakis
          Ph.D. 2009, McGill University
    - Tet Yeap
      Ph.D. 1991, University of Toronto
      - Yeap
        - Guy Ferland
          Ph.D. 2001, University of Ottawa
        - Jongsoo Choi
          Ph.D. 2006, University of Ottawa
        - Jianping Deng
          Ph.D. 2007, University of Ottawa
        - Bin Hou
          Ph.D. 2007, Beijing University of Post and Telecomm
        - Dafu Lou
          Ph.D. 2008, University of Ottawa
    - Neils da Vitoria Lobo
      Ph.D. 1992, University of Toronto
      - da Vitoria Lobo
        - Jim X. Chen
          Ph.D. 1995, University of Central Florida
          - Chen
            - Jingfang Wang
              Ph.D. 1998, George Mason University
            - Kenneth L. Alford
              Ph.D. 2000, George Mason University

- Mylopoulos
  - Tsotos
    - da Vitoria Lobo
      - Chen
        - Ying Zhu
          Ph.D. 2000, George Mason University
          - Zhu
            - Anthony S. Aquilo
              Ph.D. 2006, Georgia State University
            - Jason A. Pamplin
              Ph.D. 2007, Georgia State University
            - Jeffrey W. Chastine
              Ph.D. 2007, Georgia State University
        - Xiaodong Fu
          Ph.D. 2000, George Mason University
        - Yonggao Yang
          Ph.D. 2002, George Mason University
        - Xusheng Wang
          Ph.D. 2003, George Mason University
        - Duncan McPherson
          Ph.D. 2003, George Mason University
        - Shuangbao Wang
          Ph.D. 2004, George Mason University
        - Jayfus Doswell
          Ph.D. 2005, George Mason University
        - Fahad Alotaiby
          Ph.D. 2005, George Mason University
        - Tianshu Zhou
          Ph.D. 2008, George Mason University
        - Yanling Liu
          Ph.D. 2008, George Mason University
      - Niels Haering
        Ph.D. 1999, University of Central Florida
      - Zarina Myles
        Ph.D. 2004, University of Central Florida
      - Raymond Paul Smith
        Ph.D. 2005, University of Central Florida
    - David Wilkes
      Ph.D. 1994, University of Toronto
    - Gilbert Verghese
      Ph.D. 1995, University of Toronto
    - Yiming Ye
      Ph.D. 1997, University of Toronto
    - Neil D.B. Bruce
      Ph.D. 2008, York University
  - Hector Levesque
    Ph.D. 1981, University of Toronto
  - Levesque
    - Peter Patel-Schneider
      Ph.D. 1987, University of Toronto

- Mylopoulos
  - Levesque
    - Gerhard Lakemeyer
      Ph.D. 1990, University of Toronto
      - Lakemeyer
        - Henrik Grosskreutz
          Ph.D. 2001, RWTH Aachen University
        - Vazha Amiranashvili
          Ph.D. 2007, RWTH Aachen University
        - Alexander Ferrein
          Ph.D. 2007, RWTH Aachen University
        - Guenter Gans
          Ph.D. 2008, RWTH Aachen University
    - Yves Lesperance
      Ph.D. 1991, University of Toronto
    - Bart Selman
      Ph.D. 1990, University of Toronto
      - Selman
        - Yi-Cheng Huang
          Ph.D. 2002, Cornell University
        - Wei Wei
          Ph.D. 2005, Cornell University
        - Ioannis Vetsikas
          Ph.D. 2005, Cornell University
        - Yannet Interian
          Ph.D. 2006, Cornell University
    - Dale Eric Schuurmans
      Ph.D. 1996, University of Toronto
      - Schuurmans
        - Daniel Lizotte
          Ph.D. 2008, University of Alberta
        - Qin Iris Wang
          Ph.D. 2008, University of Alberta
        - Adam Milstein
          Ph.D. 2008, University of Waterloo
        - Feng Jiao
          Ph.D. 2008, University of Waterloo
        - Dana Wilkinson
          Ph.D. 2007, University of Waterloo
        - Yuhong Guo
          Ph.D. 2007, University of Alberta
        - Tao Wang
          Ph.D. 2007, University of Alberta
        - Linli Xu
          Ph.D. 2007, University of Waterloo
        - Jiayuan Huang
          Ph.D. 2007, University of Waterloo
        - Ali Ghodsi
          Ph.D. 2006, University of Waterloo

- Mylopoulos
  - Levesque
    - Schuurmans
      - Relu Patrascu
        Ph.D. 2004, University of Waterloo
      - Finnegan Southey
        Ph.D. 2004, University of Waterloo
      - Fletcher Lu
        Ph.D. 2003, University of Waterloo
      - Fuchun Peng
        Ph.D. 2003, University of Waterloo
    - David Mitchell
      Ph.D. 2001, University of Toronto
    - Iluju Kiringa
      Ph.D. 2003, University of Toronto
    - Steven Shapiro
      Ph.D. 2004, University of Toronto
    - Sebastian Sardina
      Ph.D. 2005, University of Toronto
    - Mikhail Soutchanski
      Ph.D. 2005, University of Toronto
    - Ronald Petrick
      Ph.D. 2006, University of Toronto
    - Yongnei Liu
      Ph.D. 2006, University of Toronto
  - Harry K.T. Wong
    Ph.D. 1983, University of Toronto
  - Michele Pilote
    Ph.D. 1983, University of Toronto
  - Sol Greenspan
    Ph.D. 1984, University of Toronto
  - James Delgrande
    Ph.D. 1985, University of Toronto
    - Delgrande
      - Diana Cukierman
        Ph.D. 2003, Simon Fraser University
      - Jens Happe
        Ph.D. 2005, Simon Fraser University
      - Aaron Hunter
        Ph.D. 2006, Simon Fraser University
  - Bryan Kramer
    Ph.D. 1986, University of Toronto
  - Lawrence K. Chung
    Ph.D. 1993, University of Toronto
    - Nary Subramia
      Ph.D. 2003, University of Texas, Dallas
  - Seshan Ananthanarayanan
    Ph.D. 1992, University of Toronto

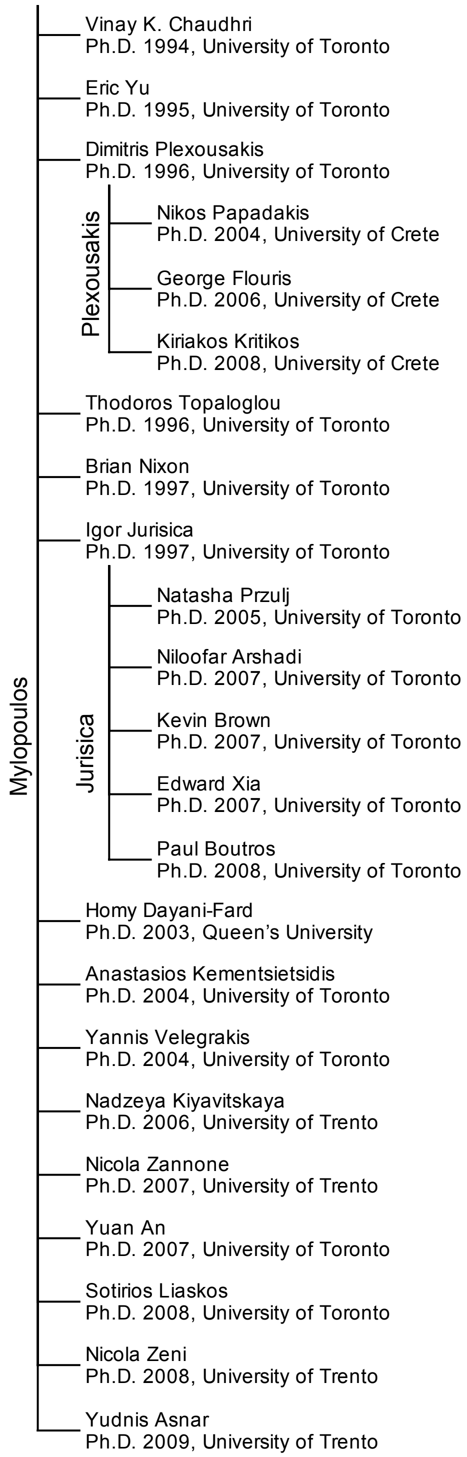

Mylopoulos
Vinay K. Chaudhri
Ph.D. 1994, University of Toronto
Eric Yu
Ph.D. 1995, University of Toronto
Dimitris Plexousakis
Ph.D. 1996, University of Toronto
Plexousakis
Nikos Papadakis
Ph.D. 2004, University of Crete
George Flouris
Ph.D. 2006, University of Crete
Kiriakos Kritikos
Ph.D. 2008, University of Crete
Thodoros Topaloglou
Ph.D. 1996, University of Toronto
Brian Nixon
Ph.D. 1997, University of Toronto
Igor Jurisica
Ph.D. 1997, University of Toronto
Jurisica
Natasha Przulj
Ph.D. 2005, University of Toronto
Niloofar Arshadi
Ph.D. 2007, University of Toronto
Kevin Brown
Ph.D. 2007, University of Toronto
Edward Xia
Ph.D. 2007, University of Toronto
Paul Boutros
Ph.D. 2008, University of Toronto
Homy Dayani-Fard
Ph.D. 2003, Queen's University
Anastasios Kementsietsidis
Ph.D. 2004, University of Toronto
Yannis Velegrakis
Ph.D. 2004, University of Toronto
Nadzeya Kiyavitskaya
Ph.D. 2006, University of Trento
Nicola Zannone
Ph.D. 2007, University of Trento
Yuan An
Ph.D. 2007, University of Toronto
Sotirios Liaskos
Ph.D. 2008, University of Toronto
Nicola Zeni
Ph.D. 2008, University of Trento
Yudnis Asnar
Ph.D. 2009, University of Trento

Table of Contents

John Mylopoulos: Sewing Seeds of Conceptual Modelling

Michael L. Brodie

Verizon Services Operations,
117 West Street,
Waltham, MA 02451-1128, USA
michael.brodie@verizon.com

1 Sewing Seeds of Conceptual Modeling

In the summer of 1980 high in the Colorado Rockies the mountain flowers were blooming, as were ideas of multi-disciplinary conceptual modelling. The Pingree Park Workshop on Data Abstraction, Database, and Conceptual Modelling [17] marked a figurative and literal high point in expectations for the exchange between databases, programming languages, and artificial intelligence (AI) on conceptual modelling. Quietly hiking amongst the AI, database, and programming language luminaries such as Ted Codd, Mike Stonebraker, Mary Shaw, Stephen Zilles, Pat Hayes, Bob Balzer, and Peter Deutsch was John Mylopoulos, a luminary himself who inspired and guided, for over three decades, the branch of Conceptual Modelling chronicled in this paper.

John's thoughtful and insightful path started in AI, reached into databases and programming languages, and then into its more natural home at the time, software engineering. In this chapter, I chronicle the leadership provided by John to the field of conceptual modeling. I argue that the conceptual modeling work started thirty years ago under John's guidance has had far reaching impact on the research in software engineering as well as to its practice in the industry. Conceptual modeling will bloom within the next decade in the form of higher-level programming across all of Computer Science.

2 A Golden Age in Computer Science

The 1970's and 1980's marked a golden age of innovation and invention in programming languages, databases, and AI. In programming languages, most of the fundamental programming paradigms used today were invented in that period, including structured [4], object-oriented, and logic programming, as well as the C family. Similarly in databases, the fundamental data models used today were invented then. The Relational Data Model[3] and the Entity Relationship (ER) model[7] transformed databases and data modelling from hierarchical and linked data structures to higher level data abstractions that in turn sparked innovation throughout the 1970's and 1980's in data abstraction and semantic data models. AI was emerging from the First AI Winter into a spring of innovation. Semantic nets, which had emerged in the 1950's and 1960's [1][2] in psychology and language translation, re-emerged as a focus of AI research in the 1980's leading to knowledge revolution into expert systems, knowledge-based systems, and knowledge engineering. At this one remarkable time the AI, database, and programming language worlds were all in a golden age of innovation and invention. While modelling was already an interest in each area, this period marked a Golden Age in the emergence of conceptual modelling due to a confluence of several important developments.

3 The Emergence of Data Modelling

The 1970's were a period of major growth in automation of business applications that led to a dramatic growth in data and transaction volumes that has continued ever since consistently exceeding Moore's Law. This growth was spurred by both demand – the need to automate, and supply – the emergence of database technology. This growth drove the need for data management across an increasingly larger range of business applications and the requirement to model increasingly complex business entities and processes. Database technology became one of the most successful technologies and the bedrock of modern business applications. Yet at the time, the physical and logical database concepts were just emerging.

Prior to the 1970's databases managed data stores of data records and focused on optimizing physical aspects of data storage and retrieval. The emergence of databases as a core tool of business applications led to the need to model business entities as real world business entities rather than mere data records. A basic principle of the emerging database discipline was shared data – a database would manage data for multiple applications. Not only must business entities be modeled in terms of business as opposed to storage, the business entities must be understandable by multiple, possibly unanticipated, applications. Another core database principle was persistence, namely that databases would represent business entities over long periods of time. The requirements to represent business entities logically rather than physically so that they could be shared by multiple applications over long periods of time led to a new research area called data modelling and the search for more expressive data models. How should business entities be modeled? How much real world information should be represented? Where should data modelers turn for inspiration?

In the 1970's, the database world advanced data modelling from the physical level to a more logical level. Since the mid1950's data had been modeled in hierarchically linked data records that represented logical parent:child or 1:N relationships to take advantage of the underlying hierarchical data structures used for storage and retrieval. In 1965 the CODASYL model was introduced to represent more complex data N:M relationships amongst records. While this model increased the expressive power of relationships, it retained many physical aspects. In 1969 Ted Codd made a Turing Award level break-through with the Relational Data Model[3] with which entities and relationships were represented as values in tables, eliminating most physical details and raising data modelling to a new, higher logical level. This was rapidly followed by the ER Model [7] that explicitly permits the modelling of business entities and the relationships amongst them. The relational data model rapidly became adopted as the dominant database model used in the vast majority of database in use today and the ER model became the dominant basis for data modelling, however data modelling at the ER level is practiced by less than 20% of the industrial database world[30], for reasons discussed later.

As the 1970's ended we had an explosion of demand for data management and data modelling, a move towards more logical data models, the emergence of data modelling, and the belief by many data modelers that more expressive models were required. So began a multi-disciplinary pursuit of modeling that had seeds at the Department of Computer Science (DCS) at the University of Toronto (UofT).

4 Seeds of Conceptual Modelling at UofT

The Golden Age in Computer Science was well under way at UofT in all areas of computing including AI, databases, and programming languages. John Mylopoulos was building an AI group, Dennis Tsichritzis was building a database group, and Jim Horning led the programming language / software engineering group. All three research groups were amongst the top five research groups in their respective areas. The multi-disciplinary direction at the university was already active in DCS across these groups in sharing courses, students, and ideas, focusing largely on modelling.

This is when John's passion for modelling, his deep insights, and his quiet guidance took root first at DCS and then beyond. From 1975 to 1980 John supervised many PhDs on modelling topics that drew on AI, databases, and programming languages, or more precisely software engineering. Nick Rossopoulos's 1975 thesis defined one of the first semantic data models[5], which was presented at the International Conference on Very Large Databases in its first session that focused on data modeling and that introduced the ER model[7] as well as two other approaches to conceptual modeling [8][9].

Michael Brodie's 1978 thesis applied programming language data types and AI logical expressions to enhance database semantic integrity augmented by AI modelling concepts to enhance the expressiveness of database schemas and databases. Dennis Tsichritzis, John Mylopoulos, and Jim Horning jointly supervised the work with additional guidance from Joachim Schmidt, the creator of Pascal-R, one of the first database programming languages (DB-PL).

Sol Greenspan's 1980 thesis applied techniques from knowledge representation to formalize the semantics of the SADT software engineering modelling methodology. The resulting paper [16] received the 10-year best paper award due to its adoption by the object-oriented community as a basis for formalizing object-oriented analysis that lead to UML.

While John made many more such contributions, such as Taxis[19], a model for information systems design, and others described elsewhere in this book, the seeds that he sewed in the late 1970's led to a wave of multi-disciplinary modelling efforts in the 1980's beyond UofT.

5 Conceptual Modelling in AI, DB, and PL

The programming language community was first to reach out to the database community to investigate the applicability programming language data type and data structures to data modeling in databases[11]. The leading candidate to share with databases was the programming language notion of data abstraction that came out of structured programming [4] and manifested in abstract data types[5] and that led to object-orientation. This interaction contributed to the Smith's aggregation – generalization data model[12][13] and data modelling methods [14] that were widely accepted in the database community.

The success of the DB-PL interactions on data abstraction and the AI-DB-PL work at UofT, inspired by John Mylopoulos, contributed to the belief that AI, database, and programming languages had mutual interests in data types, data abstraction, and data modelling. This led John Mylopoulos, Michael Brodie, and Joachim Schmidt to hold a series of workshops and to initiate a Springer Verlag book series entitled Topics in Information Systems both dedicated to exploring concepts, tools, and techniques for modelling data in AI (knowledge representation), databases (data modelling), and programming languages (data structures/programming).

The Pingree Park Workshop on Data Abstraction, Databases, and Conceptual Modelling was the highlight of the series. Innovative and influential researchers from AI, databases, and programming languages came with high expectations of mutually beneficial results. The workshop provided area overviews focusing on data modelling aspects and initiated multi-disciplinary debates on challenging issues such as data types, constraints, consistency, behavior (process) vs data, and reasoning in information systems. The proceedings were jointly published in SIGART, SIGPLAN, and SIGMOD[17].

The Intervale workshop on data semantics moved beyond Pingree Park Worshop's focus on data types as means of data modelling, data structuring, and knowledge representation, to comparing AI, database, and programming language models and methods for addressing data semantics in information systems. The results of the workshop[18] were more applicable to and had a greater impact in the AI and databases than they did in programming languages. Logic programming and datalog were introduced in this discussion and was pursued in a later, related, and similarly multi-disciplinary workshop[21].

The Islamorada Workshop Large Scale Knowledge Base and Reasoning Systems[20] extended the Pingree discussion on data types, and the Intervale discussion on modelling data semantics to conceptual modelling in the large - comparing AI knowledge base management systems with database systems[23] and addressed modelling and reasoning in large scale systems.

6 The Contributions of Early Conceptual Modelling

The conceptual modelling workshops and book series contributed to developments in all three areas: semantic data models in databases; object-orientation and UML in programming languages; and knowledge representations such as description logics in AI. Yet, as we will see later, conceptual modelling had a more natural home in software engineering, where John Mylopoulos had sewn conceptual modelling seeds that flourished for two decades. But again more on that later.

While attempts were made in the database community to investigate the potential of abstract data types[15] for modelling and correctness in databases, data types and abstract data types did not gain a footing in database management systems. The DB-PL research domain continued with the annual International Workshop on Database Programming Languages continuing to this day. Similarly databases and database abstractions did not gain a foothold in programming languages. One measure of the successful adoption of a technology is whether the technology is crosses the chasm[25], i.e., adopted by more than the "innovators" and early adopters who constitute less than 15% of the relevant market. In fact, to this day even ER modelling is not widespread in industrial database design [30].

There was a resurgence of interests in abstract data types, and data types in databases in the late 1980's that led to object-oriented databases. The debate that ensued [24] argued the challenges of implementing and using object-orientation in database systems based on the history of relational database management systems. Object-oriented databases died as a research area, but some aspects of objects were incorporated into the object-relational model. IBM made a large investment to incorporate object-relational characteristics into their flagship DBMS, DB2. The systems work required to modify DB2 was enormous and few DB2 customers ever used the object-relational features, just as predicted [24].

A Holy Grail of computing is higher level programming to provide humans with models that are more relevant to the problem domain at hand and to raise the level of modelling and interaction so that, to use IBM's famous motto, people can think and computers can work; and as Ted Codd said for the relational model, to provide greater scope for database and systems optimization by the database management system. So why would conceptual modelling not be adopted by the database world?

My experience with over 1,500 DBAs in a large enterprise and in the broader world of enterprise databases suggests a clear answer. Database design constitutes less than 1% of the database life cycle. Databases tend to be designed over a period of months and then operated for years, sometimes 30, or 40 years. ER modelling is used by a small percentage of practical database designers largely as a tool for analysis and documentation. Once the database design is approved it is compiled into relational tables. Thereafter there is no connection between the ER-based design and the relational tables. During essentially the full life of the database, DBAs must deal with the tables. Databases evolve rapidly in industry. Hence, soon after the database is compiled it is enhanced at the table level and is no longer consistent with the original ER design, had there been one. If, however, the relational tables were kept exactly in sync with the higher-level model so that any changes to one was reflect equivalently in the other, often called "round-trip engineering", the story would be much different. There are additional reasons why conceptual models have not been adopted in the mainstream database industry. The world of Telecommunications billing is extremely complex with literally thousands of features, regulatory rules, banking and credit rules, telecommunications services, choices, and packages. Not only do these features change on a daily basis, the nature of the telecommunications industry and technology leads to fundamental changes in the billing for services. Billing databases are enormous, live for decades, and contain a telecommunication organization's crown jewels. Large telecommunication organizations have many billing systems (hundreds is not uncommon) that must be integrated to provide integrated billing. And there are 1,000s of Telcos. A similar story could be told in ten other areas of telecommunications and in virtually every other industry. ER or conceptual models simply do not (yet) address these large-scale, industrial modelling issues, and if they did, their lack of round-trip engineering would significantly limit their utility.

A recurring lesson in computer science, that has been reinforced in conceptual modeling, is one that originated in philosophy and was adopted by psychology (associative memory), and later language translation[1][2] and reasoning [10] – namely that knowledge can probably be represented using nodes and links or semantic nets as they were originally called in AI. The conceptual modelling work surveyed above has contributed to the development of the node-link model in several areas. While possibly not motivated by those roots, the database world produced many node-link-based conceptual and semantic data models, the most predominant being Chen's ER model[7]. Chen's model has been the most widely adopted to date probably due to its simplicity and tractability. Yet the ER model lacks the expressive power to address the above modeling challenges of the data in telecommunication organizations. A far more expressive node-link-based model is description logics from AI, yet it poses usability issues for industrial database designers. Another area of resurgence of node-link-based knowledge representation is the semantic web. While the first stage of the semantic web was dominated by complex ontologies[28], there is a movement to adopt a far simpler model for augmenting Web resources with meta data, called the Open Links Data[29], which Tim Berners-Lee, the inventor of the web and the co-inventor of the Semantic Web, views as the future direction of the semantic web. The lesson here is not so only the recurrence of the node-link-based model, but also that "A little semantics goes a long way."[1]

7 Conceptual Modelling in Software Engineering and Beyond

The enduring discussion on data and process modelling that started in the 1970's was really between the database and the AI communities[22], sparked and nurtured by John Mylopoulos. John had a deep understanding of the conceptual modeling challenges and opportunities as well as a catholic knowledge of computer science. While his background was in AI, he also understood programming languages from his studies at Princeton, and was present at the birth of relational databases in the 1970's. For the decades from 1979 to 2009 John was key to most of the developments in conceptual modelling either directly as a contributor or indirectly as a mentor and connector across communities – across AI, database, and software engineering communities, and across various AI factions, for example, Europe vs. North America or description logics vs. datalog.

John also realized that it was the software engineering community that focused on the initial design and modelling stage of the database life cycle. It is also concerned with logical and physical requirements, specification of integrity and data quality, and the evolution of data, process and other models. Indeed, data modelling is now considered a software engineering activity rather than a database activity, as data modelling is an integral component with process modelling in the information systems life cycle.

John pursued conceptual modelling as a software engineering activity in the early 1980's when he supervised PhD theses[16][19] that contributed to mainstream

[1] Profound and now famous remark by Jim Hendler, Tetherless World Chair of Computer Science, Rensselaer Polytechnic Institute.
http://www.cs.rpi.edu/~hendler/LittleSemanticsWeb.html

software engineering such as the languages and methods surrounding UML. Hence, the software engineering community became the beneficiary of conceptual modelling and extended it to address software engineering issues, discussed elsewhere in this volume.

Now the story gets better as John Mylopoulos probably realized long ago. To return to the programming Holy Grail, humans should use high-level representations that permit them to understand the system in logical, human terms as opposed to machine level terms. Higher-level representations enable better design, analysis, monitoring, adaptation, and manipulation. Not only are higher-level representations more understandable by humans, they are also less error prone and lead to considerably higher productivity.

The challenge in achieving higher-level programming is to map the higher-level representations onto machine level representations precisely (i.e., the same semantics), equivalently (modifications in one map to semantically equivalent changes in the other), and in ways that are optimal and scalable as the system evolves in capability and grows in data and transaction volumes.

In 2001 OMG launched the Model-driven architecture (MDA) initiative to strive towards this long sought after Holy Grail. MDA is a software engineering approach for information systems development that uses models to guide systems architecture design and implementation with the objective of developing and maintaining a direct connection between the high-level model and the executable representations, to achieve the desired round-trip engineering. But you need more than a direct connection, i.e., equivalence between the high- and low-level models. Information systems evolve rapidly. Hence, changes to the high-level model, required to meet changing logical requirements must be reflected in the low-level model and changes in the low level model for optimization must be reflected equivalently in the high-level model. This capability is called agile or adaptive software development.

MDA and agile software development objectives are becoming adopted in industry with projections that initial results will be ready for industrial use in 2012. For example, Microsoft has announced support of MDA by Oslo[30] that is a forthcoming model-driven application platform.

Once MDA and agile software development are in industrial use, the entire systems life cycle can operate simultaneously at two levels – high-level models for human understanding, analysis, and modification and the executable level. This will address the lack of round trip engineering that limit the utility of today's modelling systems. At that point the modelling concepts that initiated with Conceptual Modelling, many inspired directly or indirectly by John Mylopoulos, will be directly usable across the life cycle and computing will move to a higher level – to domain models such as Telecom billing and airline reservations - and these models will be constructed with concepts, tools, and techniques that evolved from the seeds sewn in the Golden Age on computing. This will bring models and modelling to a more professional level in which models are developed by modelling experts and are standardized for reuse in the respective industry to address many of today's integration and semantic challenges.

8 And Beyond That

For three decades John Mylopoulos quietly inspired generations of researchers in the ways of conceptual modelling from his base at the University of Toronto. His vision, persistence, and insight quietly directed theses, researchers, and indeed the conceptual modelling area with contributions to AI, databases, programming languages, and software engineering. This too will get better. John is continuing his path from a new base in the mountains of Trento, Italy and soon the results of his efforts – the flowers from the seeds sewn over the three decades – will be accessible to all of computing and modelling itself will become a professional domain with John as one of its major contributors.

I am grateful for John's friendship, wisdom, and his quiet way of being – open and willing to talk and inspire us all.

References

[1] Collins, A.M., Quillian, M.R.: Retrieval time from semantic memory. Journal of verbal learning and verbal behavior 8(2), 240–248 (1969)

[2] Collins, A.M., Quillian, M.R.: Does category size affect categorization time? Journal of verbal learning and verbal behavior 9(4), 432–438 (1970)

[3] Codd, E.F.: A Relational Model of Data for Large Shared Data Banks. Commun. ACM 13(6), 377–387 (1970)

[4] Dahl, O.-J., Dijkstra, E.W., Hoare, C.A.R.: Structured Programming. Academic Press, London (1972)

[5] Liskov, B., Zilles, S.N.: Programming with Abstract Data Types. SIGPLAN Notices (SIGPLAN) 9(4), 50–59 (1974)

[6] Roussopoulos, N., Mylopoulos, J.: Using Semantic Networks for Database Management. In: VLDB 1975, pp. 144–172 (1975)

[7] Chen, P.P.: The Entity-Relationship Model: Toward a Unified View of Data. In: VLDB 1975, p. 173 (1975)

[8] Navathe, S.B., Fry, J.P.: Restructuring for Large Data Bases: Three Levels of Abstraction. In: VLDB 1975, p. 174 (1975)

[9] Senko, M.E.: Specification of Stored Data Structures and Desired Output Results in DIAM II with FORAL. In: VLDB 1975, pp. 557–571 (1975)

[10] Collins, A.M., Loftus, E.F.: A spreading-activation theory of semantic processing. Psychological Review 82(6), 407–428 (1975)

[11] Organic, E.I.: Proceedings of the 1976 Conference on Data, Abstraction, Definition and Structure, SIGPLAN Notices, Salt Lake City, Utah, United States, March 22 - 24, vol. 11(2) (1976)

[12] Smith, J.M., Smith, D.C.P.: Database Abstractions: Aggregation and Generalization. ACM Trans. Database Syst. (TODS) 2(2), 105–133 (1977)

[13] Smith, J.M., Smith, D.C.P.: Database Abstractions: Aggregation. Commun. ACM (CACM) 20(6), 405–413 (1977)

[14] Smith, J.M., Smith, D.C.P.: Principles of Database Conceptual Design. Data Base Design Techniques I, 114–146 (1978)

[15] Brodie, M.L., Schmidt, J.W.: What is the Use of Abstract Data Types? In: VLDB 1978, pp. 140–141 (1978)

[16] Greenspan, S.J., Mylopoulos, J., Borgida, A.: Capturing More World Knowledge in the Requirements Specification. In: ICSE 1982, pp. 225–235 (1980)

[17] Brodie, M.L., Zilles, S.N. (eds.): Proceedings of the Workshop on Data Abstraction, Databases and Conceptual Modelling, Pingree Park, Colorado, June 23-26 (1980); SIGART Newsletter 74 (January 1981), SIGMOD Record 11(2) (February 1981), SIGPLAN Notices 16(1) (January 1981) ISBN 0-89791-031-1
[18] Brodie, M.L., Mylopoulos, J., Schmidt, J.W. (eds.): On Conceptual Modelling, Perspectives from Artificial Intelligence, Databases, and Programming Languages, Book resulting from the Intervale Workshop 1982, Topics in Information Systems. Springer, Heidelberg (1984)
[19] Mylopoulos, J., Borgida, A., Greenspan, S.J., Wong, H.K.T.: Information System Design at the Conceptual Level - The Taxis Project. IEEE Database Eng. Bull. 7(4), 4–9 (1984)
[20] Brodie, M.L., Mylopoulos, J. (eds.): On Knowledge Base Management Systems: Integrating Artificial Intelligence and Database Technologies, Book resulting from the Islamorada Workshop 1985, Topics in Information Systems. Springer, Heidelberg (1986)
[21] Schmidt, J.W., Thanos, C. (eds.): Foundations of Knowledge Base Management: Contributions from Logic, Databases, and Artificial Intelligence, Book resulting from the Xania Workshop 1985. Topics in Information Systems. Springer, Heidelberg (1989)
[22] Brodie, M.L., Mylopoulos, J. (eds.): Readings in Artificial Intelligence and Databases. Morgan Kaufmann, San Mateo (1989)
[23] Brodie, M., Mylopoulos, J.: Knowledge Bases and Databases: Current Trends and Future Directions. In: Karagiannis, D. (ed.) IS/KI 1990 and KI-WS 1990. LNCS, vol. 474. Springer, Heidelberg (1991)
[24] Stonebraker, M., Rowe, L.A., Lindsay, B., Gray, J., Carey, M., Brodie, M., Bernstein, P., Beech, D.: Third Generation Data Base System Manifesto" (with). ACM SIGMOD Record 19(3) (September 1990)
[25] Crossing the Chasm: Marketing and Selling High-tech Products to Mainstream Customers (1991, revised 1999) ISBN 0-06-051712-3
[26] Stonebraker, M., Moore, D.: Object-Relational DBMSs: The Next Great Wave. Morgan Kaufmann, San Francisco (1996)
[27] The history of conceptual modeling, `http://cs-exhibitions.uni-klu.ac.at/index.php?id=185`
[28] OWL Web Ontology Language Reference, W3C Recommendation (February 10, 2004), `http://www.w3.org/TR/owl-ref/`
[29] Berners-Lee, T., et al.: Linked Data: Principles and State of the Art, keynote. In: 17th International World Wide Web Conference, Beijing, China, April 23-24 (2008)
[30] Hammond, J.S., Yuhanna, N., Gilpin, M., D'Silva, D.: Market Overview: Enterprise Data Modeling: A Steady State Market Prepares to Enter A transformational New Phase. In: Forrester Research, October 17 (2008)

Foundations of Temporal Conceptual Data Models

Alessandro Artale and Enrico Franconi

Faculty of Computer Science, Free University of Bozen-Bolzano, Italy
{artale,franconi}@inf.unibz.it

Abstract. This chapter considers the different temporal constructs appeared in the literature of temporal conceptual models (*timestamping* and *evolution constraints*), and it provides a coherent model-theoretic formalisation for them. It then introduces a correct and succinct encoding in a subset of first-order temporal logic, namely $\mathcal{DLR}_{\mathcal{US}}$ – the description logic $\mathcal{DLR}$ extended with the temporal operators *Since* and *Until*. At the end, results on the complexity of reasoning in temporal conceptual models are presented.

1 Introduction

Conceptual data models describe an application domain in a declarative and reusable way while constraining the use of the data by understanding what can be drawn from it. A number of conceptual modelling languages has emerged as de facto standards; in particular, we mention entity-relationship (ER) for the relational data model, UML and ODMG for the object-oriented data model, and RDF and OWL for the web ontology languages.

We consider here conceptual modelling languages able to represent dynamic and evolving information in the context of temporal databases [21, 26, 27, 32, 33, 38–40]. We provide in this chapter a mathematical foundation for them by summarising the various efforts appeared in the literature [4, 7, 23, 34, 35]. The main temporal modelling constructs we analyse can be distinguished in two main categories, *timestamping* and *evolution constraints*. To support *timestamping*, the data model should distinguish between temporal and atemporal modelling constructors; this is usually realised by a temporal marking of classes, relationships and attributes that translates into a timestamping mechanism in the corresponding database. A data model supports *evolution* constraints if it is able to keep track of how the domain elements evolve along time. In particular, *status classes* describe how elements of classes change their status from being a potential member till they cease forever to be member of the class; *transitions* deal with the fact that an object may migrate from one class to another one; while *generation constraints* describe processes that are responsible for the creation/disappearance of objects from classes.

The formalisation is based on a model-theoretic semantics that captures the meaning of both timestamping and evolution constraints. The semantics is obtained as a temporal extension of the model-theoretic semantics associated to conceptual models [14, 18]. The advantage of associating a set-theoretic semantics to a language is not only to clarify the meaning of the language constructors but also to give a semantic definition to relevant modelling notions. In particular, we are able to give a rigorous definition to the notions of: *schema satisfiability* – when a schema admits a non empty interpretation which guarantees that the constraints expressed by the schema are not contradictory; *class* and *relationships satisfiability* – when a class or a relation admits at least an interpretation in which it is not empty; *logical implication* when a new (temporal) constraint is necessarily true in a schema even if not explicitly mentioned; and finally the special case of logical implication involving *subsumption* between classes (resp. relationships) – when the interpretation of a class (resp. relationship) is necessarily a subset of the interpretation of another class (resp. relationship).

Building on the provided model-theoretic semantics we provide a correspondence between temporal conceptual models and logical theories expressed in a fragment of first order temporal logic, namely as a Description Logics (DLs) theory. DLs allow for the logical reconstruction and the extension of conceptual models (see [9, 14, 19]). The advantage of using a DL to formalise a conceptual data model lies basically on the fact that complete logical reasoning can be employed using an underlying DL inference engine to verify a conceptual specification, to infer implicit facts and stricter constraints, and to manifest any inconsistencies during the conceptual design phase. In addition, given the high complexity of the modelling task when complex data are involved, there is the demand for more sophisticated and expressive languages than for normal databases. Again, DL research is very active in providing more expressive languages for conceptual modelling (see [13, 14, 17, 18, 18, 19, 24, 31]).

In this context, we consider the temporal description logic $\mathcal{DLR}_{\mathcal{US}}$ [5], a combination of the expressive and decidable description logic $\mathcal{DLR}$ [17] (a description logic with n-ary relationships) with the linear temporal logic with temporal operators *Since* ($\mathcal{S}$) and *Until* ($\mathcal{U}$) which can be used in front of both classes and relations. We use $\mathcal{DLR}_{\mathcal{US}}$ both to capture the temporal modelling constructors in a succinct way, and to use reasoning techniques to check satisfiability, subsumption and logical implication. The mapping towards DLs presented in this chapter builds on top of a mapping which has been proved correct in [3, 4] while complexity results and algorithmic techniques can be found in [1, 5, 11]. Even if full $\mathcal{DLR}_{\mathcal{US}}$ is undecidable we address interesting modelling scenarios where subsets of the full $\mathcal{DLR}_{\mathcal{US}}$ logic is needed and where reasoning becomes a decidable problem.

The chapter is organised as follows. Section 2 describes the temporal constructs that will be the subject of the formalisation. Section 3 shows the modelling requirements that lead us to elaborate the rigorous definition of the framework presented here. Section 4 introduces the model-theoretic semantics and the notions of satisfiability, subsumption and logical implication for temporal conceptual models. The two Sections 5, 6 are the core sections where we describe how timestamping and evolution constraints can be formalised. After presenting the $\mathcal{DLR}_{\mathcal{US}}$ logic in Section 7 we proceed with a $\mathcal{DLR}_{\mathcal{US}}$ encoding of the

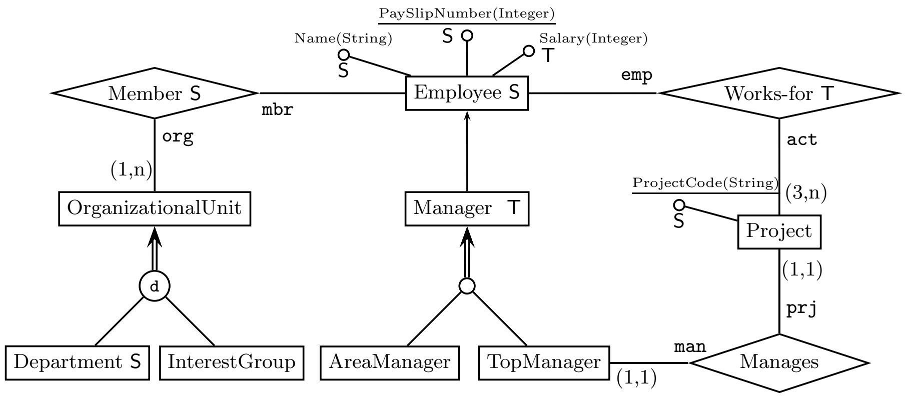


Fig. 1. The Company example

various temporal constructs in Section 8. Section 9 investigates the complexity of reasoning over temporal conceptual models and presents various scenarios where sound, complete and terminating procedures can be used. In Section 10 we state our final remarks.

2 Temporal Modelling Constructors

Temporal constructs are usually added to the classical constructs to capture the temporal behaviour of the different components of a conceptual schema. In this chapter we distinguish them in two generic classes: *Timestamping* and *Evolution* constructs.

Timestamping. It is concerned with the discrimination at the schema level between those elements of the model that change over time and others that are time invariant. Timestamping applies to classes, relationships and attributes. Data models should allow for both temporal and atemporal modelling constructors. Timestamping for attributes allows keeping how an attribute of a given object changes over time. For example (see Figure 1), the salary of an employee *emp-123* has value "*2.5K $*" for the period from 01/2004 to 12/2005, then "*3.0K $*" from 01/2006 to 12/2007, then "*3.2K $*" from 01/2008 to 12/2009.

Similarly, temporal periods can characterise an object or relationship instance as a whole rather than through its attributes. Membership in a class (relationship) can be characterised as limited in time or, vice versa, global—possibly modelling legacy classes (relationships). For example, the company schema (Figure 1) models the membership of objects in the `Employee` class as time-invariant while objects in the `Manager` class have a limited lifespan as member of that class. Timestamping is the basis for associating the notion of *lifecycle* [37] to the object/relationship instances as members of a given class/relationship (more details are given in Section 6.1). Section 5 shows how evolution constraints can be formalised.

Evolution Constraints. They control the mechanism that rules dynamic aspects, i.e., what are the permissible transitions from one state of the database to the next one [6, 7, 22, 38]. When applied to classes we talk about *Object Migration*, i.e., the evolution of an object from being member of a class to being member of another class [29]. For example, an object in the `Employee` class may migrate to become an object of the `Manager` class or an object of the `AreaManager` class can evolve into a `TopManager` class. When object migration combines with timestamping we talk about *Status Classes*. In this case we specify constraints on the membership of an object in a class by splitting it into periods according to a given classification criterion. For example, existence of a manager object in the `Manager` class can include periods where the object is an active member of the class (e.g., the manager is currently on payroll), periods where its membership is suspended (e.g., the manager is on temporary leave), and a period where its membership is disabled (e.g., the manager has left the company) [22]. The notion of status for classes allows also for a fine grained notion of lifecycle which can now depend on the membership to a particular status of a class.

Evolution-related knowledge may be conveyed also through relationships. *Generation relationships* [28] between objects of class `A` and objects of class `B` (possibly equal to `A`) describe the fact that objects in `B` are generated by objects in `A`. For example, in a company database, the splitting of a department translates into the fact that the original department generates two (or more) new departments. Clearly, if `A` and `B` are temporal classes, a generation relationship with source `A` and target `B` entails that the lifecycle of a `B` object cannot start before the lifecycle of the related `A` object. This particular temporal framework, where related objects do not coexist in time, is a form of, so called, *across-time relationships* [7, 38]. Section 6 shows how timestamping can be formalised.

In the conceptual modelling literature, different notion of 'time' have been considered. Notably, the most relevant distinction is between the so called *valid time*—which is the time when a property holds, i.e., it is true in the representation of the world—and *transaction time*—which records the history of database states rather than the world history, i.e., it is the time when a fact is *current* in the database and can be retrieved. In the following, we will consider both timestamping and evolution constructs as ranging over the valid time dimension.

3 Modelling Requirements

This Section briefly illustrates the requirements that are frequently advocated in the literature on temporal data models when dealing with temporal constraints [34, 38].

– Orthogonality. Temporal constructors should be specified separately and independently for classes, relationships, and attributes. Depending on application requirements, the temporal support must be decided by the designer.
– Upward Compatibility. This term denotes the capability of preserving the non-temporal semantics of conventional (legacy) conceptual schemas when embedded into temporal schemas.

– Snapshot Reducibility. Snapshots of the database described by a temporal schema are the same as the database described by the same schema, where all temporal constructors are eliminated and the schema is interpreted atemporally. Indeed, this property specifies that we should be able to fully rebuild a temporal database by starting from the single snapshots.

These requirements are not so obvious when dealing with evolving objects. The formalisation carried out in this chapter provides a data model able to respect these requirements also in presence of evolving objects. In particular, orthogonality affects mainly timestamping [37] and our formalisation satisfies this principle by introducing temporal marks that could be used to specify the temporal behaviour of classes, relationships, and attributes in an independent way (see Section 5). Upward compatibility and snapshot reducibility [34] are strictly related. Considered together, they allow to preserve the meaning of atemporal constructors. In particular, the meaning of classical constructors must be preserved in such a way that a designer could either use them to model classical databases, or when used in a genuine temporal setting their meaning must be preserved at each instant of time. We enforce upward compatibility by using global timestamps over legacy constructors (see Section 5). Snapshot reducibility is hard to preserve when dealing with generation relationships where involved object may not coexist. We enforce snapshot reducibility by a particular treatment of relationship typing (see Section 6.3).

4 A Formalisation of Temporal Data Models

To give a formal foundation to temporal conceptual models we briefly describe here how to associate a textual syntax to a generic EER/UML modelling language. Having a textual syntax at hand will facilitate the association of a model-theoretic semantics. In the next sections we will take advantage of such a model-theoretic temporal semantics to formally describe the temporal constructs we are interested in.

We consider a temporal conceptual model over a finite alphabet, $\mathcal{L}$, partitioned into the sets: $\mathcal{C}$ (*class* symbols), $\mathcal{A}$ (*attribute* symbols), $\mathcal{R}$ (*relationship* symbols), $\mathcal{U}$ (*role* symbols), and $\mathcal{D}$ (*domain* symbols). We consider n-ary relationships where roles from the $\mathcal{U}$ alphabet are used to distinguish the different components of a relationship, i.e., an n-ary relationship, R, connecting the (not necessarily distinct) classes $C_1, \ldots, C_n$, is defined as $R = \langle U_1 : C_1, \ldots, U_n : C_n \rangle$. Standard EER/UML constructs can also be textually defined, like *Attributes* for both classes and relationships (we use the notation $\textsc{att}(C) = \langle A_1 : D_1, \ldots, A_h : D_h \rangle$ to denote all the attributes of a class C, and similarly for attributes of relationships); *Participation Constraints* denoting the cardinality in the participation of a class into a relationship; *Isa* for both classes and relationships (denoted as $C_1 \textsc{isa} C_2$ or $R_1 \textsc{isa} R_2$, respectively); *Disjointness* and *Covering* constraints over a class hierarchy. For a complete set of EER/UML constructs and their textual definition we refer to [3, 4, 19].

In Figure 1 we show our running example of an EER schema for a company database where classes and relationships are denoted by boxes and diamonds, respectively; directed arrows stand for isa; double arrows denote a covering constraint; a circled ‘d’ denotes a disjoint hierarchy; participation constraints are indicated with numbers in round brackets; timestamps are denoted with S (snapshot) and T (temporary).

The model-theoretic semantics that gives a foundation to temporal modelling languages adopts the so called *snapshot*[1] representation of abstract temporal databases and temporal conceptual models [20]. Following the snapshot paradigm, relations of a temporal database are interpreted by a mapping function depending on a specific point in time. The flow of time $\mathcal{T} = \langle \mathcal{T}_p, < \rangle$, where $\mathcal{T}_p$ is a set of time points (or chronons) and $<$ is a binary precedence relation on $\mathcal{T}_p$, is assumed to be isomorphic to either $\langle \mathbb{Z}, < \rangle$ or $\langle \mathbb{N}, < \rangle$. Thus, a standard relational database can be regarded as the result of mapping a temporal database from a specific time point in $\mathcal{T}$ to an atemporal database, with the assumption that the interpretation of both constants and the domain are invariant over time.

Definition 1 (Temporal Schemas Semantics). *Let Σ be a temporal schema. A* temporal database state *for the schema Σ is a tuple $\mathcal{B} = (\mathcal{T}, \Delta^{\mathcal{B}} \cup \Delta^{\mathcal{B}}_D, \cdot^{\mathcal{B}(t)})$, such that: $\Delta^{\mathcal{B}}$ is a nonempty set of abstract objects disjoint from $\Delta^{\mathcal{B}}_D$; $\Delta^{\mathcal{B}}_D = \bigcup_{D_i \in \mathcal{D}} \Delta^{\mathcal{B}}_{D_i}$ is the set of basic domain values used in the schema Σ; and $\cdot^{\mathcal{B}(t)}$ is a function that for each $t \in \mathcal{T}$ maps:*

- *Every basic domain symbol D_i into a set $D_i^{\mathcal{B}(t)} = \Delta^{\mathcal{B}}_{D_i}$.*
- *Every class C to a set $C^{\mathcal{B}(t)} \subseteq \Delta^{\mathcal{B}}$.*
- *Every relationship R to a set $R^{\mathcal{B}(t)}$ of $\mathcal{U}$-labeled tuples over $\Delta^{\mathcal{B}}$—i.e., let $R = \langle U_1 : C_1, \ldots, U_n : C_n \rangle$ be an n-ary relationship connecting the classes $C_1, \ldots, C_n$, then, $\forall t \in \mathcal{T}. \forall r \in R^{\mathcal{B}(t)} \rightarrow (r = \langle U_1 : o_1, \ldots, U_n : o_n \rangle \wedge \forall i \in \{1, \ldots, n\}. o_i \in C_i^{\mathcal{B}(t)})$. We adopt the convention: $\langle U_1 : o_1, \ldots, U_n : o_n \rangle \equiv \langle o_1, \ldots, o_n \rangle$, when $\mathcal{U}$-labels are clear from the context.*
- *Every attribute A to a set $A^{\mathcal{B}(t)} \subseteq \Delta^{\mathcal{B}} \times \Delta^{\mathcal{B}}_D$, such that, for each $C \in \mathcal{C}$, if $\text{ATT}(C) = \langle A_1 : D_1, \ldots, A_h : D_h \rangle$, then, $\forall t \in \mathcal{T}. \forall o \in C^{\mathcal{B}(t)} \rightarrow (\forall i \in \{1, \ldots, h\}, \forall a_i. \langle o, a_i \rangle \in A_i^{\mathcal{B}(t)} \rightarrow a_i \in D_i^{\mathcal{B}(t)})$.*

$\mathcal{B}$ is said a legal temporal database state *if it satisfies all of the constraints expressed in the schema*[2].

Given such a set-theoretic semantics we are able to rigorously define some relevant modelling notions such as satisfiability, subsumption and derivation of new constraints by means of logical implication.

Definition 2. *Let Σ be a schema, $C \in \mathcal{C}$ a class, and $R \in \mathcal{R}$ a relationship. The following modelling notions can be defined:*

[1] The snapshot model represents the same class of temporal databases as the so called *timestamp* model [33, 34] which adds a temporal attribute to each relation [20].

[2] We don’t report here the semantics for temporal constraints since they will be discussed in details in the next Sections. As for the semantics of participation constraints, isa, disjointness and covering constraints we refer to [4].

1. *C (R) is* satisfiable *if there exists a legal temporal database state $\mathcal{B}$ for Σ such that $C^{\mathcal{B}(t)} \neq \emptyset$ ($R^{\mathcal{B}(t)} \neq \emptyset$), for some $t \in \mathcal{T}$;*
2. *Σ is* satisfiable *if there exists a legal temporal database state $\mathcal{B}$ for Σ such that at least one class of Σ is not empty ($\mathcal{B}$ is also said a* model *for Σ);*
3. *C_1 (R_1) is* subsumed *by C_2 (R_2) in Σ if every legal temporal database state for Σ is also a legal temporal database state for $C_1 \text{ISA} C_2$ ($R_1 \text{ISA} R_2$);*
4. *A schema Σ' is* logically implied *by a schema Σ over the same alphabet if every legal temporal database state for Σ is also a legal temporal database state for Σ'.*

5 Timestamping

A temporal model supports *timestamping* if it is able to distinguish between *snapshot* constructors—i.e., constructors with a global lifespan associated to each of their instances—*temporary* constructors—i.e., each of their instances has a limited lifespan—or *mixed* constructors—i.e., their instances can have either a global or a temporary existence. In the following, a class, relationship or attribute is called temporal if it is either temporary or mixed. The two temporal marks, S (snapshot) and T (temporary), introduced at the conceptual level (see Figure 1), together with unmarked constructors capture the temporal distinction between snapshot, temporary and mixed constructors. Notice that, the temporal behaviour of an attribute can be either *globally* forced, or *locally* defined when associated to single classes. Since the local constraint is more general we assume that attributes are locally temporally constrained. At the end of this section we also introduce two notions strictly related to timestamping: that one of a (temporal) key, and a variant of participation constraints called *lifespan participation constraints*. We now proceed with the semantics of timestamping that can be defined as follows (not quantified variables are assumed to be universally quantified):

$o \in C^{\mathcal{B}(t)} \rightarrow \forall t' \in \mathcal{T}. o \in C^{\mathcal{B}(t')}$	`Snapshot Class`
$o \in C^{\mathcal{B}(t)} \rightarrow \exists t' \neq t. o \notin C^{\mathcal{B}(t')}$	`Temporary Class`
$r \in R^{\mathcal{B}(t)} \rightarrow \forall t' \in \mathcal{T}. r \in R^{\mathcal{B}(t')}$	`Snapshot Relationship`
$r \in R^{\mathcal{B}(t)} \rightarrow \exists t' \neq t. r \notin R^{\mathcal{B}(t')}$	`Temporary Relationship`
$(o \in C^{\mathcal{B}(t)} \wedge \langle o, a_i \rangle \in A_i^{\mathcal{B}(t)}) \rightarrow \forall t' \in \mathcal{T}. \langle o, a_i \rangle \in A_i^{\mathcal{B}(t')}$	`Snapshot Attribute`
$(o \in C^{\mathcal{B}(t)} \wedge \langle o, a_i \rangle \in A_i^{\mathcal{B}(t)}) \rightarrow \exists t' \neq t. \langle o, a_i \rangle \notin A_i^{\mathcal{B}(t')}$	`Temporary Attribute`

The following "classical" desirable features found in the literature of temporal conceptual modelling come as almost trivial logical implications form the above formalisation.

Proposition 1. (Timestamps: Logical Implications [4]) *In every temporal schema supporting timestamping, the following temporal properties hold:*

1. *Subclass of temporary classes are also temporary (similarly for relationships).*
2. *If exactly one of a whole set of snapshot subclasses partitioning*[3] *a snapshot superclass is temporary, then, the whole set of classes is unsatisfiable.*
3. *Participants of snapshot relationships are either snapshot or unmarked classes.*
4. *Participants of snapshot relationships are snapshot when they participate at least once in the relationship.*
5. *A relationship is temporary if one of the participating classes is temporary.*

On the other hand, nothing can be said about subclasses of snapshot or unmarked classes and classes participating to temporary or unmarked relationships: they can be snapshot, temporary, or unmarked classes. Since the domain of an attribute is not restricted to the classes they are attached to, the temporal behaviour of a class is independent of that of its attributes.

Example 1. Considering our running example of Figure 1, the following logical implications hold:

– Because `Manager` is a temporary class, then both `AreaManager` and `TopManager` are temporary classes; constraining either `AreaManager` or `TopManager` as snapshot classes would lead to a contradiction. On the other hand, even if `Employee` is a snapshot class, it is consistent to have `Manager`—a temporary class—as a subclass of `Employee`.
– Because `OrganizationalUnit` participates at least once in a snapshot relationship, then, it must be a snapshot class.
– Since `InterestGroup` participates in a partition of snapshot classes it must be a snapshot class, too.
– The fact that `Manages` must be a temporary relationship follows logically from our theory because the temporary class `TopManager` participates in the relationship.
– Since the temporal behaviour of classes is independent from that one of its attributes, the fact the `Salary` is a temporary attribute of the snapshot class `Employee` is admitted.

Key Constraints. As a byproduct of attribute timestamping we can define single-attribute keys (visualised in a schema as an underlined attribute, e.g., `PaySlipNumber` is a key for the class `Employee`) as a mandatory and single-valued snapshot attribute that uniquely identifies objects of the class. Assuming that A_{key} is a key for the class C, then the the following formalisation holds:

$$(o \in C^{\mathcal{B}(t)} \wedge \langle o, a_{key}\rangle \in A_{key}^{\mathcal{B}(t)}) \rightarrow \forall t' \in \mathcal{T}.\langle o, a_{key}\rangle \in A_{key}^{\mathcal{B}(t')} \quad \texttt{Snapshot Attribute}$$
$$o \in C^{\mathcal{B}(t)} \rightarrow \exists^{=1} a_{key} \in \Delta_D^{\mathcal{B}}.\langle o, a_{key}\rangle \in A_{key}^{\mathcal{B}(t)} \quad \texttt{Mandatory \& Single-Valued}$$
$$a_{key} \in \Delta_D^{\mathcal{B}} \rightarrow \exists^{\leq 1} o \in C^{\mathcal{B}(t)}.\langle o, a_{key}\rangle \in A_{key}^{\mathcal{B}(t)} \quad \texttt{Uniqueness}$$

[3] The partition must be a disjoint covering.

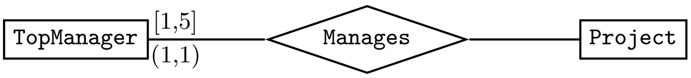


Fig. 2. Lifespan and "classical" participation constraints

Lifespan Participation Constraints. While classical participations constraints are evaluated at each point in time *lifespan participation constraints* (represented in a schema by a pair of values in square brackets) [26, 37, 39] are evaluated during the entire existence of the object. Notice that, since the set of instances of snapshot relationships does not change in time, there is no difference between "classical" and lifespan participation constraints for snapshot relationships. For example, if we want to state that a top manager should manage at most five different projects in his entire existence while still being constrained in managing exactly one project at a time, we can use a combination of the two participation constraints (see Figure 2). The model-theoretic semantics for lifespan participation is the following:

$$o \in C^{\mathcal{B}(t)} \rightarrow k \leq \# \bigcup_{t' \in \mathcal{T}} \{r \in R^{\mathcal{B}(t')} \mid r[U] = o\} \leq m \quad \texttt{Lifespan Participation Constraint}$$

6 Evolution Constraints

Evolution constraints contribute in modelling the temporal dynamic of an object. In this section we propose a formalisation of the basic temporal concepts that are at the root of advanced conceptual temporal models: *status classes*, distinguished in scheduled, active, suspended and disabled; *transitions* of objects between different classes; *generation* relationships asserting evolution constraints on objects linked by temporal relationships. In this section we aim at presenting a formal characterisation of the temporal conceptual modelling constructors capturing the evolution of objects.

6.1 Status Classes

The *Status* [7, 22, 37] is a conceptual notion associated to temporal classes to rule the lifecycle of their objects. It records the evolving state of membership of each object in the class. Following [37], status modelling includes up to four different statuses, and the allowed transitions between them:

– Scheduled. An object is scheduled if the planning of its existence within the class has to be recorded while its membership in the class will only become effective (active) some time later. For example, if a new project is approved but will not start until a later date the given project can be created as a new object in the `Project` class, with status scheduled for the valid time interval starting at the date of the approval decision and ending at the expected launching date. Each scheduled object will eventually become an active object. Supporting a scheduled status avoids the introduction of a new time type, the decision time [22].

- Active. The status of an object is active if the object is a full member of the class (and therefore conforms to its type). For example, a currently ongoing project is an active member, at time now, of the `Project` class.
- Suspended. This status qualifies objects that exist as members of the class, but are to be seen as temporarily inactive members of the class [22]. An employee taking a temporary leave of absence is an example of what can be considered as a suspended employee. Only active objects can be suspended. A suspended object was in the past an active one.
- Disabled. This status is used to specify that the object's membership in the class has expired. While logically deleted, disabled objects are kept for some specific application purposes, e.g., statistical analyses. A disabled object was in the past an active member of the class (an object cannot be created in the disabled status). It can never again become a non-disabled member of that class (e.g., an expired project cannot be reactivated).

Let C be a temporal (i.e., temporary or mixed) class. We capture status transition of membership in C by associating to C the following *status classes*: `Scheduled-C`, `Suspended-C`, `Disabled-C`. In particular, status classes are constrained by the hierarchy of Figure 3 (where C may also be mixed) that classifies C instances according to their actual status. To preserve upward compatibility we do not explicitly introduce an active class, but assume by default that the name of the class itself denotes the set of active objects, i.e., `Active-C` $\equiv$ `C`. Note that, since membership of objects into snapshot classes is global, i.e., objects are always active, the notion of status classes does not apply to snapshot classes.

To capture the intended meaning of status classes, we define ad-hoc constraints and then show that such constraints capture the evolving behaviour of status classes as described in the literature [22, 37]. First of all, disjointness and isa constraints between statuses of a class C can be described as illustrated in

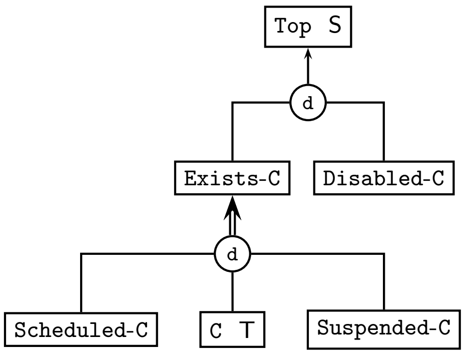


Fig. 3. Status classes

Figure [3], where `Top` is supposed to be a snapshot class which represents the universe of abstract objects (i.e., $\texttt{Top}^{\mathcal{B}(t)} \equiv \Delta^{\mathcal{B}}$). Other than hierarchical constraints, the intended semantics of status classes induces the following rules that are related to their temporal behaviour:

(Exists) *Existence persists until Disabled.*
$o \in \texttt{Exists-C}^{\mathcal{B}(t)} \rightarrow \forall t' > t.(o \in \texttt{Exists-C}^{\mathcal{B}(t')} \vee o \in \texttt{Disabled-C}^{\mathcal{B}(t')})$
(Disab1) *Disabled persists.*
$o \in \texttt{Disabled-C}^{\mathcal{B}(t)} \rightarrow \forall t' > t.o \in \texttt{Disabled-C}^{\mathcal{B}(t')}$
(Disab2) *Disabled was Active in the past.*
$o \in \texttt{Disabled-C}^{\mathcal{B}(t)} \rightarrow \exists t' < t.o \in \texttt{C}^{\mathcal{B}(t')}$
(Susp) *Suspended was Active in the past.*
$o \in \texttt{Suspended-C}^{\mathcal{B}(t)} \rightarrow \exists t' < t.o \in \texttt{C}^{\mathcal{B}(t')}$
(Sch1) *Scheduled will eventually become Active.*
$o \in \texttt{Scheduled-C}^{\mathcal{B}(t)} \rightarrow \exists t' > t.o \in \texttt{C}^{\mathcal{B}(t')}$
(Sch2) *Scheduled can never follow Active.*
$o \in \texttt{C}^{\mathcal{B}(t)} \rightarrow \forall t' > t.o \notin \texttt{Scheduled-C}^{\mathcal{B}(t')}$

As a consequence of the above formalisation the following set of new rules can be derived.

Proposition 2 (Status Classes: Logical Implications [7]). *Given a temporal schema supporting status classes, then, the following logical implications hold:*

1. *Disabled classes will never become active anymore.*
2. *The scheduled status persists until the class become active.*
3. *A scheduled class cannot evolve directly into a disabled status.*

Temporal applications often use concepts that are derived from the notion of object statuses, e.g., the *lifespan* of a temporal object or its *birth* and *death* instants. Hereinafter we provide formal definitions for these concepts.

Lifespan and related notions. The lifespan of an object w.r.t. a class describes the temporal instants where the object can be considered a member of the class. With the introduction of status classes we can distinguish between the following notions: $\textsc{ExistenceSpan}_C$, $\textsc{LifeSpan}_C$, $\textsc{ActiveSpan}_C$, $\textsc{Begin}_C$, $\textsc{Birth}_C$ and $\textsc{Death}_C$. They are functions which depend on the object membership to the status classes associated to a temporal class C.

The *existencespan* of an object describes the temporal instants where the object is either a scheduled, active or suspended member of a given class. More formally, $\textsc{ExistenceSpan}_C : \Delta^{\mathcal{B}} \rightarrow 2^{\mathcal{T}}$, such that:

$$\textsc{ExistenceSpan}_C(o) = \{t \in \mathcal{T} \mid o \in \texttt{Exists-C}^{\mathcal{B}(t)}\}$$

The *lifespan* of an object describes the temporal instants where the object is an active or suspended member of a given class (thus, $\textsc{LifeSpan}_C(o) \subseteq \textsc{ExistenceSpan}_C(o)$). More formally, $\textsc{LifeSpan}_C : \Delta^{\mathcal{B}} \rightarrow 2^{\mathcal{T}}$, such that:

$$\textsc{LifeSpan}_C(o) = \{t \in \mathcal{T} \mid o \in \mathtt{C}^{\mathcal{B}(t)} \cup \mathtt{Suspended\text{-}C}^{\mathcal{B}(t)}\}$$

The *activespan* of an object describes the temporal instants where the object is an active member of a given class (thus, $\textsc{ActiveSpan}_C(o) \subseteq \textsc{LifeSpan}_C(o)$). More formally, $\textsc{ActiveSpan}_C : \Delta^{\mathcal{B}} \to 2^{\mathcal{T}}$, such that:

$$\textsc{ActiveSpan}_C(o) = \{t \in \mathcal{T} \mid o \in \mathtt{C}^{\mathcal{B}(t)}\}$$

The functions $\textsc{Begin}_C$ and $\textsc{Death}_C$ associate to an object the first and the last appearance, respectively, of the object as a member of a given class, while $\textsc{Birth}_C$ denotes the first appearance as an active object of that class. More formally, $\textsc{Begin}_C, \textsc{Birth}_C, \textsc{Death}_C : \Delta^{\mathcal{B}} \to \mathcal{T}$, such that:

$$\begin{aligned}
&\textsc{Begin}_C(o) = \mathtt{min}(\textsc{ExistenceSpan}_C(o))\\
&\textsc{Birth}_C(o) = \mathtt{min}(\textsc{ActiveSpan}_C(o)) \equiv \mathtt{min}(\textsc{LifeSpan}_C(o))\\
&\textsc{Death}_C(o) = \mathtt{max}(\textsc{LifeSpan}_C(o))
\end{aligned}$$

We could still speak of existencespan, lifespan or activespan for snapshot classes, but in this case they all collapse to the full time line, $\mathcal{T}$. Furthermore, $\textsc{Begin}_C(o)$ $= \textsc{Birth}_C(o) = -\infty$, and $\textsc{Death}_C(o) = +\infty$ either when C is a snapshot class or in cases of instances existing since ever and/or living forever.

6.2 Transition

Transition constraints [29, 37] have been introduced to model the phenomenon called *object migration*. A transition records objects migrating from a *source* class to a *target* class. At the schema level, it expresses that the instances of the source class may *migrate* into the target class. Two types of transitions have been considered: *dynamic evolution*, when objects cease to be instances of the source class to become instances of the target class, and *dynamic extension*, when the creation of the target instance does not force the removal of the source instance. For example, considering the company schema (Figure 1), if we want to record data about the promotion of area managers into top managers we can specify a dynamic evolution from the class `AreaManager` to the class `TopManager`. We can also record the fact that a mere employee becomes a manager by defining a dynamic extension from the class `Employee` to the class `Manager` (see Figure 4). Regarding the graphical representation, as illustrated in Figure 4, we use a dashed arrow pointing to the target class and labeled with either DEX or DEV denoting dynamic extension and evolution, respectively.

Specifying a transition between two classes means that: *a)* We want to keep track of such migration; *b)* Not necessarily all the objects in the source or in the target participate in the migration; *c)* When the source class is a temporal class, migration only involves active or suspended objects—thus, neither disabled nor scheduled objects can take part in a transition.

In the following, we present a formalisation that satisfies the above requirements. We represent transitions by introducing a new class denoted by either $\textsc{dex}_{C_1,C_2}$ or $\textsc{dev}_{C_1,C_2}$ for dynamic extension and evolution, respectively. More

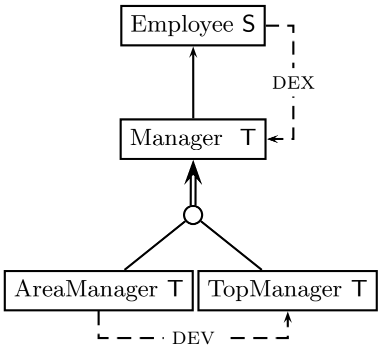


Fig. 4. Transitions employee-to-manager and area-to-top manager

formally, in case of a *dynamic extension* between classes C_1, C_2 the following semantic equation holds:

$$o \in \text{DEX}_{C_1,C_2}^{\mathcal{B}(t)} \rightarrow (o \in (\texttt{Suspended-C}_1{}^{\mathcal{B}(t)} \cup \texttt{C}_1{}^{\mathcal{B}(t)}) \wedge o \notin \texttt{C}_2{}^{\mathcal{B}(t)} \wedge o \in C_2^{\mathcal{B}(t+1)})$$

In case of a *dynamic evolution* between classes C_1, C_2 the source object cannot remain active in the source class. Thus, the following semantic equation holds:

$$\begin{aligned} o \in \text{DEV}_{C_1,C_2}^{\mathcal{B}(t)} \rightarrow (o \in (\texttt{Suspended-C}_1{}^{\mathcal{B}(t)} \cup \texttt{C}_1{}^{\mathcal{B}(t)}) \wedge o \notin \texttt{C}_2{}^{\mathcal{B}(t)} \wedge \\ o \in C_2^{\mathcal{B}(t+1)} \wedge o \notin C_1^{\mathcal{B}(t+1)}) \end{aligned}$$

Finally, we formalise the case where the source (C_1) and/or the target (C_2) totally participate in a dynamic extension/evolution (at schema level we add mandatory cardinality constraints on DEX/DEV links):

$$\begin{array}{ll} o \in C_1^{\mathcal{B}(t)} \rightarrow \exists t' > t. o \in \text{DEX}_{C_1,C_2}^{\mathcal{B}(t')} & \texttt{Source Total Transition} \\ o \in C_2^{\mathcal{B}(t)} \rightarrow \exists t' < t. o \in \text{DEX}_{C_1,C_2}^{\mathcal{B}(t')} & \texttt{Target Total Transition} \\ o \in C_1^{\mathcal{B}(t)} \rightarrow \exists t' > t. o \in \text{DEV}_{C_1,C_2}^{\mathcal{B}(t')} & \texttt{Source Total Evolution} \\ o \in C_2^{\mathcal{B}(t)} \rightarrow \exists t' < t. o \in \text{DEV}_{C_1,C_2}^{\mathcal{B}(t')} & \texttt{Target Total Evolution} \end{array}$$

An interesting set of consequences of the above proposed modelling of dynamic transitions are shown in the following proposition.

Proposition 3 (Transition: Logical Implications [7]). *Given a schema supporting transitions for objects, then the following logical implications hold:*

1. *The classes* DEX_{C_1,C_2} *and* DEV_{C_1,C_2} *are temporary classes; actually, they hold at single time points.*
2. *Objects in the classes* DEX_{C_1,C_2} *and* DEV_{C_1,C_2} *cannot be disabled as* C_2*.*
3. *The target class* C_2 *cannot be snapshot (it becomes temporary in case of both* `Source Total Transition` *and* `Target Total Evolution` *constraints).*
4. *As a consequence of dynamic evolution, the source class,* C_1*, cannot be snapshot (and it becomes temporary in case of* `Source Total Evolution` *constraints).*

5. *Dynamic evolution cannot be specified between a class and one of its sub-classes.*
6. *Dynamic extension between disjoint classes logically implies Dynamic evolution.*

6.3 Generation Relationships

Generation relationships [7, 28, 36, 37] represent processes that lead to the emergence of *new objects* starting from a set of existing objects. In their most generic form, a generation relationship can have a collection of objects as source and a collection of objects as target. For example (see Figure 5), assuming an organisation remodels its departments, it may be that an existing department is split into two new departments, while two existing departments are merged into a single new department and three existing departments are reorganised as two new departments. Cardinality constraints can be added to specify the cardinality of sets involved in a generation. For example, if we want to record the fact that a group of managers proposes at most one new project at a time a generation relationship from `Manager` to `Project` can be defined with the cardinality "*at most one*" on the manager side.

Depending whether the source objects are preserved (as member of the source class) or disabled by the generation process, we distinguish between *production* and *transformation* relationships, respectively. Managers creating projects is an example of the former, while departmental reorganisation is an example of the latter. At the conceptual level we introduce two marks for generation relationships: GP for production and GT for transformation relationships, and an arrow pointing to the target class (see Figure 5).

We model generation as binary relationships connecting a source class to a target one, with the target being in its scheduled status: $\text{REL}(R) = \langle \texttt{source} : C_1, \texttt{target} : \texttt{Scheduled-C}_2 \rangle$. The semantics of *production relationships*, R, is described by the following equation:

$$\langle o_1, o_2 \rangle \in R^{\mathcal{B}(t)} \rightarrow (o_1 \in C_1^{\mathcal{B}(t)} \wedge o_2 \in \texttt{Scheduled-C}_2{}^{\mathcal{B}(t)} \wedge o_2 \in C_2^{\mathcal{B}(t+1)})$$

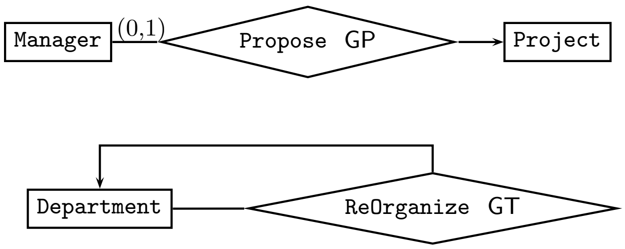


Fig. 5. Production and transformation generation relationships

Thus, objects active in the source class produce objects active in the target class at the next point in time. A production relationship is a case of across-time relationships [7]—i.e., relationships connecting objects which are active in the connected classes at different points in time—where the use of status classes allows us to preserve snapshot reducibility. Indeed, for each pair of objects, $\langle o_1, o_2 \rangle$, belonging to a generation relationships o_1 is active in the source while o_2 is scheduled in the target.

The case of *transformation* is captured by the following semantic equation:

$$\langle o_1, o_2 \rangle \in R^{\mathcal{B}(t)} \rightarrow (o_1 \in C_1^{\mathcal{B}(t)} \wedge o_1 \in \texttt{Disabled-C}_1{}^{\mathcal{B}(t+1)} \wedge \\ o_2 \in \texttt{Scheduled-C}_2{}^{\mathcal{B}(t)} \wedge o_2 \in C_2^{\mathcal{B}(t+1)})$$

Thus, objects active in the source generate objects active in the target at the next point in time while the source objects cease to exist as member of the source. As for production relationships, transformations are special cases of across-time relationships.

Proposition 4 (Generation: Logical Implications [7]). *The following logical implications hold as a consequence of the generation semantics:*

1. *A generation relationship, R, is temporary; actually, it is instantaneous.*
2. *The target class, C_2, cannot be snapshot (C_2 must be temporary if it participates at least once).*
3. *Objects participating as target cannot be disabled.*
4. *If R is a transformation relationship, then, C_1 cannot be snapshot (C_1 must be temporary if it participates at least once).*

Note that the `Department` class that is both the source and target of a transformation relationship (Figure 5) can no longer be snapshot (as it was in Example 1) and must be changed to temporary. Furthermore, as a consequence of this new timestamp for the `Department` class, `InterestGroup` is now a genuine mixed class.

Starting with the next section we provide a correspondence between temporal conceptual schemas and theories expressed in temporal description logics.

7 The Temporal Description Logic

As the description logic $\mathcal{DLR}$ has been used to reason over conceptual models [14, 17, 18] in this chapter we use a temporal extension of $\mathcal{DLR}$ to capture temporal conceptual models. The temporal description logic $\mathcal{DLR}_{\mathcal{US}}$ [5, 25] combines the propositional temporal logic with *Since* and *Until* and the (non-temporal) description logic $\mathcal{DLR}$ [13, 17]. $\mathcal{DLR}_{\mathcal{US}}$ can be regarded as a rather expressive fragment of the first-order temporal logic $L^{\{\textbf{since, until}\}}$ (cf. [20, 30]).

The basic syntactical types of $\mathcal{DLR}_{\mathcal{US}}$ are *concepts* (unary predicates) and n-ary *relations* of arity ≥ 2. Starting from a set of *atomic concepts* (denoted

$$\begin{aligned}
C \rightarrow\ & \top \mid \bot \mid CN \mid \neg C \mid C_1 \sqcap C_2 \mid \exists^{\lesseqgtr k}[U_j]R \mid \\
& \Diamond^+ C \mid \Diamond^- C \mid \Box^+ C \mid \Box^- C \mid \oplus C \mid \ominus C \mid C_1 \mathcal{U} C_2 \mid C_1 \mathcal{S} C_2 \\
R \rightarrow\ & \top_n \mid RN \mid \neg R \mid R_1 \sqcap R_2 \mid U_i/n : C \mid \\
& \Diamond^+ R \mid \Diamond^- R \mid \Box^+ R \mid \Box^- R \mid \oplus R \mid \ominus R \mid R_1 \mathcal{U} R_2 \mid R_1 \mathcal{S} R_2
\end{aligned}$$

$$\begin{aligned}
\top^{\mathcal{I}(t)} &= \Delta^{\mathcal{I}} \\
\bot^{\mathcal{I}(t)} &= \emptyset \\
CN^{\mathcal{I}(t)} &\subseteq \top^{\mathcal{I}(t)} \\
(\neg C)^{\mathcal{I}(t)} &= \top^{\mathcal{I}(t)} \setminus C^{\mathcal{I}(t)} \\
(C_1 \sqcap C_2)^{\mathcal{I}(t)} &= C_1^{\mathcal{I}(t)} \cap C_2^{\mathcal{I}(t)} \\
(\exists^{\lesseqgtr k}[U_j]R)^{\mathcal{I}(t)} &= \{\, d \in \top^{\mathcal{I}(t)} \mid \sharp\{\langle d_1, \ldots, d_n\rangle \in R^{\mathcal{I}(t)} \mid d_j = d\} \lesseqgtr k\} \\
(C_1 \mathcal{U} C_2)^{\mathcal{I}(t)} &= \{\, d \in \top^{\mathcal{I}(t)} \mid \exists v > t.(d \in C_2^{\mathcal{I}(v)} \wedge \forall w \in (t,v).d \in C_1^{\mathcal{I}(w)})\} \\
(C_1 \mathcal{S} C_2)^{\mathcal{I}(t)} &= \{\, d \in \top^{\mathcal{I}(t)} \mid \exists v < t.(d \in C_2^{\mathcal{I}(v)} \wedge \forall w \in (v,t).d \in C_1^{\mathcal{I}(w)})\}
\end{aligned}$$

$$\begin{aligned}
(\top_n)^{\mathcal{I}(t)} &\subseteq (\Delta^{\mathcal{I}})^n \\
RN^{\mathcal{I}(t)} &\subseteq (\top_n)^{\mathcal{I}(t)} \\
(\neg R)^{\mathcal{I}(t)} &= (\top_n)^{\mathcal{I}(t)} \setminus R^{\mathcal{I}(t)} \\
(R_1 \sqcap R_2)^{\mathcal{I}(t)} &= R_1^{\mathcal{I}(t)} \cap R_2^{\mathcal{I}(t)} \\
(U_i/n : C)^{\mathcal{I}(t)} &= \{\, \langle d_1, \ldots, d_n\rangle \in (\top_n)^{\mathcal{I}(t)} \mid d_i \in C^{\mathcal{I}(t)}\} \\
(R_1 \mathcal{U} R_2)^{\mathcal{I}(t)} &= \{\, \langle d_1, \ldots, d_n\rangle \in (\top_n)^{\mathcal{I}(t)} \mid \\
&\qquad \exists v > t.(\langle d_1, \ldots, d_n\rangle \in R_2^{\mathcal{I}(v)} \wedge \forall w \in (t,v).\, \langle d_1, \ldots, d_n\rangle \in R_1^{\mathcal{I}(w)})\} \\
(R_1 \mathcal{S} R_2)^{\mathcal{I}(t)} &= \{\, \langle d_1, \ldots, d_n\rangle \in (\top_n)^{\mathcal{I}(t)} \mid \\
&\qquad \exists v < t.(\langle d_1, \ldots, d_n\rangle \in R_2^{\mathcal{I}(v)} \wedge \forall w \in (v,t).\, \langle d_1, \ldots, d_n\rangle \in R_1^{\mathcal{I}(w)})\} \\
(\Diamond^+ R)^{\mathcal{I}(t)} &= \{\langle d_1, \ldots, d_n\rangle \in (\top_n)^{\mathcal{I}(t)} \mid \exists v > t.\, \langle d_1, \ldots, d_n\rangle \in R^{\mathcal{I}(v)}\} \\
(\oplus R)^{\mathcal{I}(t)} &= \{\langle d_1, \ldots, d_n\rangle \in (\top_n)^{\mathcal{I}(t)} \mid \langle d_1, \ldots, d_n\rangle \in R^{\mathcal{I}(t+1)}\} \\
(\Diamond^- R)^{\mathcal{I}(t)} &= \{\langle d_1, \ldots, d_n\rangle \in (\top_n)^{\mathcal{I}(t)} \mid \exists v < t.\, \langle d_1, \ldots, d_n\rangle \in R^{\mathcal{I}(v)}\} \\
(\ominus R)^{\mathcal{I}(t)} &= \{\langle d_1, \ldots, d_n\rangle \in (\top_n)^{\mathcal{I}(t)} \mid \langle d_1, \ldots, d_n\rangle \in R^{\mathcal{I}(t-1)}\}
\end{aligned}$$

Fig. 6. Syntax and semantics of $\mathcal{DLR}_{\mathcal{US}}$

by CN), a set of *atomic relations* (denoted by RN), and a set of *role symbols* (denoted by U) we hereinafter define inductively (complex) concepts and relation expressions as is shown in the upper part of Figure 6, where the binary constructors ($\sqcap, \sqcup, \mathcal{U}, \mathcal{S}$) are applied to relations of the same arity, i, j, k, n are natural numbers, $i \leq n$, and j does not exceed the arity of R.

The non-temporal fragment of $\mathcal{DLR}_{\mathcal{US}}$ coincides with $\mathcal{DLR}$. For both concept and relation expressions all the Boolean constructors are available. The selection expression $U_i/n : C$ denotes an n-ary relation whose argument named U_i ($i \leq n$) is of type C; if it is clear from the context, we omit n and write $(U_i : C)$. The projection expression $\exists^{\lesseqgtr k}[U_j]R$ is a generalisation with cardinalities of the projection operator over the argument named U_j of the relation R; the plain classical projection is $\exists^{\geq 1}[U_j]R$ (we will use $\exists[U_j]R$ as a shortcut). It is also possible to use the pure argument position version of the language by replacing role symbols U_i with the corresponding position numbers i. To show the expressive power of $\mathcal{DLR}_{\mathcal{US}}$ we refer to the next sections where $\mathcal{DLR}_{\mathcal{US}}$ is used to capture various forms of temporal constraints.

The model-theoretic semantics of $\mathcal{DLR}_{\mathcal{US}}$ assumes a flow of time $\mathcal{T} = \langle \mathcal{T}_p, < \rangle$, where $\mathcal{T}_p$ is a set of time points (or chronons) and $<$ a binary precedence relation on $\mathcal{T}_p$, is assumed to be isomorphic to $\langle \mathbb{Z}, < \rangle$. The language of $\mathcal{DLR}_{\mathcal{US}}$ is interpreted in *temporal models* over $\mathcal{T}$, which are triples of the form $\mathcal{I} \doteq \langle \mathcal{T}, \Delta^{\mathcal{I}}, \cdot^{\mathcal{I}(t)} \rangle$, where $\Delta^{\mathcal{I}}$ is non-empty set of objects (the *domain* of $\mathcal{I}$) and $\cdot^{\mathcal{I}(t)}$ an *interpretation function* such that, for every $t \in \mathcal{T}$ (in the following the notation $t \in \mathcal{T}$ is used as a shortcut for $t \in \mathcal{T}_p$), every concepts C, and every n-ary relation R, we have $C^{\mathcal{I}(t)} \subseteq \Delta^{\mathcal{I}}$ and $R^{\mathcal{I}(t)} \subseteq (\Delta^{\mathcal{I}})^n$. The semantics of concept and relation expressions is defined in the lower part of Figure 6, where $(u, v) = \{w \in \mathcal{T} \mid u < w < v\}$. For concepts, the temporal operators $\diamond^+$ (some time in the future), $\oplus$ (at the next moment), and their past counterparts can be defined via $\mathcal{U}$ and $\mathcal{S}$: $\diamond^+ C \equiv \top \mathcal{U} C$, $\oplus C \equiv \bot \mathcal{U} C$, etc. The operators $\Box^+$ (always in the future) and $\Box^-$ (always in the past) are the duals of $\diamond^+$ (some time in the future) and $\diamond^-$ (some time in the past), respectively, i.e., $\Box^+ C \equiv \neg \diamond^+ \neg C$ and $\Box^- C \equiv \neg \diamond^- \neg C$, for both concepts and relations. The operators $\diamond^*$ (at some moment) and its dual $\Box^*$ (at all moments) can be defined for both concepts and relations as $\diamond^* C \equiv C \sqcup \diamond^+ C \sqcup \diamond^- C$ and $\Box^* C \equiv C \sqcap \Box^+ C \sqcap \Box^- C$, respectively.

A *knowledge base*, $\mathcal{K}$, is a finite set of $\mathcal{DLR}_{\mathcal{US}}$ axioms of the form $C_1 \sqsubseteq C_2$ and $R_1 \sqsubseteq R_2$, with R_1 and R_2 being relations of the same arity. The notation $C_1 \doteq C_2$ ($R_1 \doteq R_2$) is a shortcut for $C_1 \sqsubseteq C_2$, $C_2 \sqsubseteq C_1$ ($R_1 \sqsubseteq R_2$, $R_2 \sqsubseteq R_1$). An interpretation $\mathcal{I}$ satisfies $C_1 \sqsubseteq C_2$ ($R_1 \sqsubseteq R_2$) if and only if the interpretation of C_1 (R_1) is included in the interpretation of C_2 (R_2) at all time, i.e., $C_1^{\mathcal{I}(t)} \subseteq C_2^{\mathcal{I}(t)}$ ($R_1^{\mathcal{I}(t)} \subseteq R_2^{\mathcal{I}(t)}$), for all $t \in \mathcal{T}$. Various *reasoning services* can be defined in $\mathcal{DLR}_{\mathcal{US}}$. A knowledge base, $\mathcal{K}$, is *satisfiable* if there is an interpretation that satisfies all the axioms in $\mathcal{K}$ (in symbols, $\mathcal{I} \models \mathcal{K}$). A knowledge base, $\mathcal{K}$, *logically implies* an axiom, $C_1 \sqsubseteq C_2$ ($R_1 \sqsubseteq R_2$), and write $\mathcal{K} \models C_1 \sqsubseteq C_2$ ($\Sigma \models R_1 \sqsubseteq R_2$), if we have $\mathcal{I} \models C_1 \sqsubseteq C_2$ ($\mathcal{I} \models R_1 \sqsubseteq R_2$) whenever $\mathcal{I} \models \mathcal{K}$. In this latter case, the concept C_1 (relation R_1) is said to be *subsumed* by the concepts C_2 (relation R_2) in the knowledge base $\mathcal{K}$. A concepts C is satisfiable, given a knowledge base $\mathcal{K}$, if there exists a model $\mathcal{I}$ of $\mathcal{K}$ such that $C^{\mathcal{I}(t)} \neq \emptyset$ for some $t \in \mathcal{T}$, i.e., $\mathcal{K} \not\models C \sqsubseteq \bot$. A relation R is satisfiable, given a knowledge base $\mathcal{K}$, if there exists a model $\mathcal{I}$ of $\mathcal{K}$ such that $R^{\mathcal{I}(t)} \neq \emptyset$ for some $t \in \mathcal{T}$, i.e., $\mathcal{K} \not\models R \sqsubseteq \bot$. Finally, knowledge base satisfiability, concepts subsumption and relation satisfiability can be reduced to concepts satisfiability in the following way: $\mathcal{K} \not\models \top \sqsubseteq \bot$, $\mathcal{K} \models C_1 \sqcap \neg C_2 \sqsubseteq \bot$, $\mathcal{K} \not\models \exists^{\geq 1}[U_j]R \sqsubseteq \bot$ for some $j \leq n$ where n is the arity of R, respectively.

While $\mathcal{DLR}$ knowledge bases are fully able to capture atemporal EER/UML schemas [14, 17, 18]—i.e., given an EER schema there is an equi-satisfiable $\mathcal{DLR}$ knowledge base—in the following sections we use $\mathcal{DLR}_{\mathcal{US}}$ knowledge bases to capture temporal EER/UML schemas with both timestamping and evolution constraints.

8 Encoding Temporal Schemas in Description Logics

We start by briefly summarising how knowledge bases in the description logic $\mathcal{DLR}$ can capture conceptual schemas. The correspondence we report here is based on a mapping introduced by [14, 18, 19] for atemporal EER models.

Informally, the encoding works as follows. Class and relationship symbols in a conceptual diagram are mapped into $\mathcal{DLR}$ concept names and relation names (with the same arity of the original relationship), respectively. Domain symbols are mapped into additional concept names, pairwise disjoint. Attributes of classes are mapped to binary relation names in $\mathcal{DLR}$ with number restrictions stating the cardinality of the attribute to distinguish between single- and multi-valued attributes. *Isa* links between classes or between relationships are mapped using $\mathcal{DLR}$ axioms. Generalised hierarchies with disjointness and covering constraints can be captured using Boolean connectives. Cardinality constraints are mapped using number restriction in $\mathcal{DLR}$.

Let us consider the class diagram depicted in Figure 1 and representing (a portion of) a company database. According to the diagram, all managers are employees and are partitioned into area managers and top managers. This information can be represented by means of the following $\mathcal{DLR}$ axioms:

$$\begin{aligned}
\texttt{Manager} &\sqsubseteq \texttt{Employee}\\
\texttt{AreaManager} &\sqsubseteq \texttt{Manager}\\
\texttt{TopManager} &\sqsubseteq \texttt{Manager}\\
\texttt{AreaManager} &\sqsubseteq \neg\texttt{TopManager}\\
\texttt{Manager} &\sqsubseteq \texttt{AreaManager} \sqcup \texttt{TopManager}
\end{aligned}$$

Binary relation names of the form $A \sqsubseteq \texttt{From}:\top \sqcap \texttt{To}:\top$ capture attributes. Each employee has three functional attributes (by default, we assume that attributes are single-valued and mandatory), `Salary`, `PaySlipNumber`, with integer values, and `Name`, with string values; here we show only the first:

$$\texttt{Employee} \sqsubseteq \exists^{=1}[\texttt{from}]\texttt{Salary} \sqcap \exists^{=1}[\texttt{from}](\texttt{Salary} \sqcap \texttt{to}/2 : \texttt{Integer})$$

The binary relationship `Works-for` has `Employee`s as domain, while the range is restricted to `Project`s:

$$\texttt{Works-for} \sqsubseteq \texttt{emp}/2 : \texttt{Employee} \sqcap \texttt{act}/2 : \texttt{Project}$$

Each top manager manages exactly one project, while a project must involve at least three employees:

$$\begin{aligned}
\texttt{TopManager} &\sqsubseteq \exists^{=1}[\texttt{man}]\texttt{Manages}\\
\texttt{Project} &\sqsubseteq \exists^{\geq 3}[\texttt{act}]\texttt{Works-For}
\end{aligned}$$

Temporal properties expressed in the diagram are mapped using temporal operators in $\mathcal{DLR}_{\mathcal{US}}$. In the following we will show how to extend the above translation in order to capture both timestamping and evolution constraints.

Encoding Timestamping

Timestamping is the ability to distinguish between *snapshot* constructors—i.e., constructors with a global lifespan associated to each of their instances—*temporary* constructors—i.e., each of their instances has a limited lifespan—or *mixed* constructors—i.e., their instances can have either a global or a temporary existence. Timestamps for both classes and relationships are captured by the following $\mathcal{DLR}_{\mathcal{US}}$ axioms (remember that $\Box^*$ is the "at all time" operator while $\Diamond^*$ is the "at some time" operator, see Section 7):

(SnapC)	$C \sqsubseteq \Box^* C$	`Snapshot Class`
(TempC)	$C \sqsubseteq \Diamond^* \neg C$	`Temporary Class`
(SnapT)	$R \sqsubseteq \Box^* R$	`Snapshot Relationship`
(TempR)	$R \sqsubseteq \Diamond^* \neg R$	`Temporary Relationship`

Considering timestamping for attributes we first recall that attributes are captured in $\mathcal{DLR}$ as binary relations. Thus, the following $\mathcal{DLR}_{\mathcal{US}}$ axioms hold[4]:

(SnapA)	$C \sqsubseteq \neg\exists[\mathtt{From}](A \sqcap \Diamond^* \neg A)$	`Snapshot Attribute`
(TempA)	$C \sqsubseteq \neg\exists[\mathtt{From}](\Box^* A)$	`Temporary Attribute`

Key Constraints. We now show the $\mathcal{DLR}_{\mathcal{US}}$ axioms that capture the notion of single-attribute keys (see the case of a pay slip number that uniquely identifies an employee in Figure 1). We need three axioms: the first to specify that a key is a *snapshot* attribute, the second to characterise a key as *mandatory* and *single-valued*, and the last axiom to specify *uniqueness*. Assuming that A_{key} is a key for the class C, then its semantics is captured by the following $\mathcal{DLR}_{\mathcal{US}}$ axioms:

(Key1)	$C \sqsubseteq \neg\exists[\mathtt{From}](A_{key} \sqcap \Diamond^* \neg A_{key})$	`Snapshot Attribute`
(Key2)	$C \sqsubseteq \exists^{=1}[\mathtt{from}]A_{key}$	`Mandatory & Single-Valued`
(Key3)	$\top \sqsubseteq \exists^{\leq 1}[\mathtt{to}](A_{key} \sqcap \mathtt{from} : C)$	`Uniqueness`

Lifespan Participation Constraints. Lifespan participation constraints (see Figure 2) are formalised in $\mathcal{DLR}_{\mathcal{US}}$ using a combination of number restrictions and temporal operators for relations:

(LPC) $C \sqsubseteq \exists^{\geq \mathtt{k}}[\mathtt{U}]\Diamond^*\mathtt{R} \sqcap \exists^{\leq \mathtt{m}}[\mathtt{U}]\Diamond^*\mathtt{R}$ `Lifespan Participation Constraint`

The standard logical implications due to timestamping and showed in Proposition 1 can be rephrased in terms of $\mathcal{DLR}_{\mathcal{US}}$ logical implications.

Proposition 5 (Timestamps: Logical Implications [4]). *In every temporal schema supporting timestamping, the following temporal properties hold:*

1. *Subclass of temporary classes are also temporary (similarly for relationships).*
   $\{C_1 \sqsubseteq C, C \sqsubseteq \Diamond^* \neg C\} \models C_1 \sqsubseteq \Diamond^* \neg C_1$
2. *If exactly one of a whole set of snapshot subclasses partitioning a snapshot superclass is temporary, then, the whole set of classes is unsatisfiable (we consider a three class partition).*
   $\{C_0 \doteq C_1 \sqcup C_2, C_1 \sqsubseteq \neg C_2, C_0 \sqsubseteq \Box^* C_0, C_1 \sqsubseteq \Box^* C_1, C_2 \sqsubseteq \Diamond^* \neg C_2\} \models C_i \sqsubseteq \bot, \quad i = 0, 1, 2$

[4] The axioms consider a local temporal behaviour for attributes. To associate a global behaviour to an attribute we consider it as a binary relationship and apply the axioms for timestamping relationships.

3. *Participants of snapshot relationships are either snapshot or unmarked classes.* $\{R \sqsubseteq \Box^* R, R \sqsubseteq U_i : C_i, C_i \sqsubseteq \Diamond^* \neg C_i\} \models R \sqsubseteq \bot$
4. *Participants of snapshot relationships are snapshot when they participate at least once in the relationship.* $\{R \sqsubseteq \Box^* R, R \sqsubseteq U_i : C_i, C_i \sqsubseteq \exists [U_i] R\} \models C_i \sqsubseteq \Box^* C_i$
5. *A relationship is temporary if one of the participating classes is temporary.* $\{R \sqsubseteq U_i : C_i, C_i \sqsubseteq \Diamond^* \neg C_i\} \models R \sqsubseteq \Diamond^* \neg R$

Encoding Status Classes

Status classes record the evolving state of membership of each object in the class. We distinguish four status: scheduled, active, suspended and disabled. $\mathcal{DLR}_{\mathcal{US}}$ axioms are able to fully capture the hierarchical constraints of Figure 3. Moreover, the semantic equations formalising status classes are captured by the following set of $\mathcal{DLR}_{\mathcal{US}}$ axioms:

(Exists)	$\texttt{Exists-C} \sqsubseteq \Box^+(\texttt{Exists-C} \sqcup \texttt{Disabled-C})$
(Disab1)	$\texttt{Disabled-C} \sqsubseteq \Box^+ \texttt{Disabled-C}$
(Disab2)	$\texttt{Disabled-C} \sqsubseteq \Diamond^- \texttt{C}$
(Susp)	$\texttt{Suspended-C} \sqsubseteq \Diamond^- \texttt{C}$
(Sch1)	$\texttt{Scheduled-C} \sqsubseteq \Diamond^+ \texttt{C}$
(Sch2)	$\texttt{C} \sqsubseteq \Box^+ \neg \texttt{Scheduled-C}$

We denote with Σ_{st} the above set of axioms together with the $\mathcal{DLR}_{\mathcal{US}}$ axioms that capture the hierarchy of Figure 3. We can now rephrase the logical implications involving status classes showed in Proposition 2 as $\mathcal{DLR}_{\mathcal{US}}$ logical implications.

Proposition 6 (Status Classes: Logical Implications [7]). *Given the set of $\mathcal{DLR}_{\mathcal{US}}$ axioms Σ_{st} that capture status classes, the following logical implications hold:*

1. *Disabled will never become active anymore.* $\Sigma_{st} \models \texttt{Disabled-C} \sqsubseteq \Box^+ \neg \texttt{C}$
2. *Scheduled persists until active.* $\Sigma_{st} \models \texttt{Scheduled-C} \sqsubseteq \texttt{Scheduled-C} \mathcal{U} \texttt{C}$
3. *Scheduled cannot evolve directly to Disabled.* $\Sigma_{st} \models \texttt{Scheduled-C} \sqsubseteq \oplus \neg \texttt{Disabled-C}$

Encoding Transition

Transition constraints model the so called object migration. They are distinguished in *dynamic evolution*—when objects cease to be instances of the source class to become instances of the target class—and *dynamic extension*—when the creation of the target instance does not force the removal of the source instance. We represent transitions by introducing a new class denoted by either DEX_{C_1,C_2} or DEV_{C_1,C_2} for dynamic extension and evolution, respectively. The $\mathcal{DLR}_{\mathcal{US}}$ axioms capturing these temporal constraints are:

(DEX) $\text{DEX}_{C_1,C_2} \sqsubseteq (\texttt{Suspended-C}_1 \sqcup \texttt{C}_1) \sqcap \neg\texttt{C}_2 \sqcap \oplus\, C_2$
(DEV) $\text{DEV}_{C_1,C_2} \sqsubseteq (\texttt{Suspended-C}_1 \sqcup \texttt{C}_1) \sqcap \neg\texttt{C}_2 \sqcap \oplus\, (C_2 \sqcap \neg C_1)$

The $\mathcal{DLR}_{\mathcal{US}}$ axioms capturing the cases where the source (C_1) and/or the target (C_2) totally participate in a dynamic extension/evolution are:

(STT)	$C_1 \sqsubseteq \diamond^+\text{DEX}_{C_1,C_2}$	`Source Total Transition`
(TTT)	$C_2 \sqsubseteq \diamond^-\text{DEX}_{C_1,C_2}$	`Target Total Transition`
(STE)	$C_1 \sqsubseteq \diamond^+\text{DEV}_{C_1,C_2}$	`Source Total Evolution`
(TTE)	$C_2 \sqsubseteq \diamond^-\text{DEV}_{C_1,C_2}$	`Target Total Evolution`

We can now rephrase the logical implications involving transition constraints showed in Proposition 3 as $\mathcal{DLR}_{\mathcal{US}}$ logical implications.

Proposition 7 (Transition: Logical Implications [7]). *Let* $\Sigma_{tr} = \{$(DEV), (DEX)$\}$*, then the following logical implications hold:*

1. *The classes* DEX_{C_1,C_2} *and* DEV_{C_1,C_2} *are temporary classes; actually, they hold at single time points.*
   $\Sigma_{st} \cup \Sigma_{tr} \models \text{DEX}_{C_1,C_2} \sqsubseteq \oplus\, \neg\text{DEX}_{C_1,C_2} \sqcap\ \ominus\, \neg\text{DEX}_{C_1,C_2}$
   $\Sigma_{st} \cup \Sigma_{tr} \models \text{DEV}_{C_1,C_2} \sqsubseteq \oplus\, \neg\text{DEV}_{C_1,C_2} \sqcap\ \ominus\, \neg\text{DEV}_{C_1,C_2}$
2. *Objects in the classes* DEX_{C_1,C_2} *and* DEV_{C_1,C_2} *cannot be disabled as* C_2*.*
   $\Sigma_{st} \cup \Sigma_{tr} \models \text{DEX}_{C_1,C_2} \sqsubseteq \neg\texttt{Disabled-C}_2$
   $\Sigma_{st} \cup \Sigma_{tr} \models \text{DEV}_{C_1,C_2} \sqsubseteq \neg\texttt{Disabled-C}_2$
3. *The target class* C_2 *cannot be snapshot (it becomes temporary in case of both* (TTT) *and* (TTE) *constraints).*
   $\Sigma_{st} \cup \Sigma_{tr} \models \text{DEX}_{C_1,C_2} \sqsubseteq \diamond^*[C_2 \sqcap (\diamond^+\neg C_2 \sqcup \diamond^-\neg C_2)]$
4. *As a consequence of dynamic evolution, the source class,* C_1*, cannot be snapshot (and it becomes temporary in case of* (STE) *constraints).*
   $\Sigma_{st} \cup \Sigma_{tr} \models \text{DEV}_{C_1,C_2} \sqsubseteq \diamond^*[C_1 \sqcap (\diamond^+\neg C_1 \sqcup \diamond^-\neg C_1)]$
5. *Dynamic evolution cannot be specified between a class and one of its subclasses.*
   $\Sigma_{st} \cup \Sigma_{tr} \cup \{C_2 \sqsubseteq C_1\} \models \text{DEV}_{C_1,C_2} \sqsubseteq \bot$
6. *Dynamic extension between disjoint classes logically implies Dynamic evolution.*
   $\Sigma_{st} \cup \Sigma_{tr} \cup \{C_1 \sqsubseteq \neg C_2\} \models \text{DEX}_{C_1,C_2} \sqsubseteq \text{DEV}_{C_1,C_2}$

Encoding Generation Relationships

Generation relationships lead to the emergence of new objects starting from a set of existing objects. Depending whether the source objects are preserved (as member of the source class) or disabled, we distinguish between *production* and *transformation* relationships, respectively. The $\mathcal{DLR}_{\mathcal{US}}$ axioms capturing the production and transformation semantics are:

(PROD) $R \sqsubseteq \texttt{source} : C_1 \sqcap \texttt{target} : (\texttt{Scheduled-C}_2 \sqcap \oplus\, C_2)$
(TRANS) $R \sqsubseteq \texttt{source} : (C_1 \sqcap \oplus\, \texttt{Disabled-C}_1) \sqcap \texttt{target} : (\texttt{Scheduled-C}_2 \sqcap \oplus\, C_2)$

We can now rephrase the logical implications involving generation relationships showed in Proposition 4 as $\mathcal{DLR}_{\mathcal{US}}$ logical implications.

Proposition 8 (Generation: Logical Implications [7]). *The following logical implications hold as a consequence of the $\mathcal{DLR}_{\mathcal{US}}$ axioms capturing generation relationships:*

1. *A generation relationship, R, is temporary; actually, it is instantaneous.*
   $\Sigma_{st} \cup \{(\textsc{Prod})\} \models R \sqsubseteq \Box^{+}\neg R \sqcap \Box^{-}\neg R$
2. *The target class, C_2, cannot be snapshot (C_2 must be temporary if it participates at least once).*
   $\Sigma_{st} \cup \{(\textsc{Prod})\} \models R \sqsubseteq \texttt{target}{:}\Diamond^{*}[C_2 \sqcap (\Diamond^{+}\neg C_2 \sqcup \Diamond^{-}\neg C_2)]$
3. *Objects participating as target cannot be disabled.*
   $\Sigma_{st} \cup \{(\textsc{Prod})\} \models R \sqsubseteq \texttt{target}{:}\neg\texttt{Disabled-C}_2$
4. *If R is a transformation relationship, then, C_1 cannot be snapshot (C_1 must be temporary if it participates at least once).*
   $\Sigma_{st} \cup \{(\textsc{Trans})\} \models R \sqsubseteq \texttt{source}{:}\Diamond^{*}[C_1 \sqcap (\Diamond^{+}\neg C_1 \sqcup \Diamond^{-}\neg C_1)]$

8.1 Correctness of the Encoding

To prove that reasoning on temporal schemas can be done by reasoning on their $\mathcal{DLR}_{\mathcal{US}}$ translation, we need to prove the correctness of the encoding. That temporal schemas with timestamping and transition constraints can be encoded as description logic theories has been proven correct in [4, 5] by establishing a precise correspondence between legal database states of temporal schemas and models of the corresponding description logic theories. This result can be easily extended to the full set of temporal constraints presented here.

Theorem 1 (Correctness of the encoding). *Let Σ be a temporal schema. Then, Σ admits a legal database state if and only if the corresponding $\mathcal{DLR}_{\mathcal{US}}$ knowledge base encoding the schema has a model.*

This characterisation allows us to support the reasoning on temporal conceptual models, as in Definition 2, by using the reasoning services of $\mathcal{DLR}_{\mathcal{US}}$. On the other hand, since reasoning with $\mathcal{DLR}_{\mathcal{US}}$ theories is undecidable, in the following section we present interesting scenarios where reasoning become decidable together with their respective complexity results.

9 Complexity of Reasoning on Temporal Models

As this chapter shows, the temporal description logic $\mathcal{DLR}_{\mathcal{US}}$ is able to fully capture temporal schemas with both timestamping and evolution constraints. On the other hand, reasoning over $\mathcal{DLR}_{\mathcal{US}}$ knowledge bases, i.e., checking satisfiability, subsumption and logical implications, turns out to be undecidable [5]. The main reason for this is the possibility to couple the evolution of concepts with the possibility to postulate that a binary relation does not vary in time (i.e., global relations). Note that, showing that temporal schemas can be mapped into $\mathcal{DLR}_{\mathcal{US}}$ axioms does not necessarily imply that reasoning over temporal schemas is an undecidable problem. Unfortunately, [1] shows that the undecidable Halting

Problem can be encoded as the problem of class satisfiability w.r.t. a temporal schema with, among the others, the following constructs: disjoint and covering constraints, sub-relationships, timestamping on both classes and relationships, and evolution constraints.

On the other hand, the fragment, $\mathcal{DLR}^{-}_{\mathcal{US}}$, of $\mathcal{DLR}_{\mathcal{US}}$ deprived of the ability to talk about temporal persistence of n-ary relations, for $n \geq 2$, is decidable. Indeed, reasoning in $\mathcal{DLR}^{-}_{\mathcal{US}}$ is an EXPTIME-complete problem [5]. This result gives us an useful scenario where reasoning over temporal schemas becomes decidable. In particular, if we forbid timestamping for relationships (i.e., relationships are just unmarked and interpreted as mixed constructors) reasoning on temporal models with just class timestamping but full evolution constraints can be reduced to reasoning over $\mathcal{DLR}^{-}_{\mathcal{US}}$. The problem of reasoning in this setting is complete for EXPTIME since the EXPTIME-complete problem of reasoning with $\mathcal{ALC}$ knowledge bases can be captured by such schemas [14].

We maintain decidability also by allowing full timestamping (i.e., timestamping for relationships, attributes and classes) but dropping evolution constraints. This is the basic temporal conceptual modelling scenario where temporal marks allow to distinguish between temporary and global constructs (this scenario also allows for both temporal keys and lifespan participation constraints). This scenario is decidable since it is possible to encode temporal schemas without evolution constraints by using a combination between the description logic $\mathcal{DLR}$ and the epistemic modal logic $\mathsf{S5}$ (see [12] for the exact mapping). Reasoning over $\mathcal{DLR}_{S5}$ has been proved to be decidable and 2-EXPTIME-complete [11] by extending a previous result on the logic $\mathcal{ALC}_{S5}$ [25].

Other interesting scenarios currently under investigation are the cases where the temporal expressivity is maintained in its full capability (i.e., both full timestamping and evolution constraints) but some of the constructs used at the conceptual level are dropped. In particular, by dropping isa between relationships and/or partitioning constraints we could regain decidability in the full temporal scenario. In this case we can use description logics from the *DL-Lite* family [2, 15, 16] to capture these weaker forms of conceptual schemas (see [8] for an exhaustive description of the data models that can be captured inside *DL-Lite*). A demoralisation of *DL-Lite* has been proposed in [10] where reasoning is showed to be EXPSPACE-complete. As a future work we plan to study the mapping of the various temporal constructs presented here in the temporal extension of *DL-Lite* and to investigate a tight complexity bound for the resulting temporal data modelling language.

10 Conclusions

This chapter summarises the various proposals appeared in the literature about temporal conceptual data models within a formal framework. We presented a model-theoretic semantics for different temporal constructs grouped along two generic categories, i.e., timestamping and evolution constraints. The given formal semantics clarifies the meaning of the modelling constructors and also gives a rigorous definition to relevant design support tasks such as satisfiability of schemas, classes and relationships; subsumption for both classes and relationships; general logical implication. Furthermore, for each constructor we have shown how desirable properties can be derived as logical implications from the proposed formalisation.

We have been able to show how temporal schemas can be equivalently expressed using a subset of first-order temporal logic, i.e., $\mathcal{DLR}_{\mathcal{US}}$, the description logic $\mathcal{DLR}$ extended with the temporal operators *Since* and *Until*. While $\mathcal{DLR}_{\mathcal{US}}$ is an undecidable language, several decidable sub-languages can be used to reason over temporal schemas. Since these sub-languages usually do not mix timestamping with evolution constraints, we are currently investigating new scenarios where, by weakening the atemporal expressiveness of the conceptual model, we regain decidability of the full temporal setting. We started to work on these encouraging scenarios by using *DL-Lite* as the atemporal DL and extending it to capture time varying domains.

Acknowledgements

We would like to thank our colleagues Roman Kontchakov, Carsten Lutz, Federica Mandreoli, Christine Parent, Vladislav Ryzhikov, Stefano Spaccapietra, David Toman, Frank Wolter, and Michael Zakharyaschev with whom we carried out most of the work presented in this chapter.

References

1. Artale, A.: Reasoning on temporal conceptual schemas with dynamic constraints. In: 11th Int. Symposium on Temporal Representation and Reasoning (TIME 2004). IEEE Computer Society, Los Alamitos (2004); also in Proc. of the 2004 Int. Workshop on Description Logics (DL 2004)
2. Artale, A., Calvanese, D., Kontchakov, R., Zakharyaschev, M.: DL-Lite in the light of first-order logic. In: Proc. of AAAI 2007, pp. 361–366 (2007)
3. Artale, A., Franconi, E.: Temporal ER modeling with description logics. In: Akoka, J., Bouzeghoub, M., Comyn-Wattiau, I., Métais, E. (eds.) ER 1999. LNCS, vol. 1728, pp. 81–95. Springer, Heidelberg (1999)
4. Artale, A., Franconi, E., Mandreoli, F.: Description logics for modelling dynamic information. In: Chomicki, J., van der Meyden, R., Saake, G. (eds.) Logics for Emerging Applications of Databases. LNCS, Springer, Heidelberg (2003)
5. Artale, A., Franconi, E., Wolter, F., Zakharyaschev, M.: A temporal description logic for reasoning over conceptual schemas and queries. In: Flesca, S., Greco, S., Leone, N., Ianni, G. (eds.) JELIA 2002. LNCS(LNAI), vol. 2424, pp. 98–110. Springer, Heidelberg (2002)
6. Artale, A., Parent, C., Spaccapietra, S.: Modeling the evolution of objects in temporal information systems. In: Dix, J., Hegner, S.J. (eds.) FoIKS 2006. LNCS, vol. 3861, pp. 22–42. Springer, Heidelberg (2006)
7. Artale, A., Parent, C., Spaccapietra, S.: Evolving objects in temporal information systems. Annals of Mathematics and Artificial Intelligence 50(1-2), 5–38 (2007)

8. Artale, A., Calvanese, D., Kontchakov, R., Ryzhikov, V., Zakharyaschev, M.: Reasoning over extended ER models. In: Parent, C., Schewe, K.-D., Storey, V.C., Thalheim, B. (eds.) ER 2007. LNCS, vol. 4801, pp. 277–292. Springer, Heidelberg (2007)
9. Artale, A., Cesarini, F., Soda, G.: Describing database objects in a concept language environment. IEEE Trans. on Knowledge and Data Engineering 8(2), 345–351 (1996)
10. Artale, A., Kontchakov, R., Lutz, C., Wolter, F., Zakharyaschev, M.: Temporalising tractable description logics. In: 14th International Symposium on Temporal Representation and Reasoning (TIME 2007). IEEE Computer Society Press, Los Alamitos (2007)
11. Artale, A., Lutz, C., Toman, D.: A description logic of change. In: Int. Joint Conference on Artificial Intelligence (IJCAI 2007), Hyderabad, India (January 2007)
12. Artale, A., Toman, D.: Decidable reasoning over timestamped conceptual models. In: Proc. of the 21st Int. Workshop on Description Logics (DL 2008), Dresden, Germany (May 2008)
13. Baader, F., Calvanese, D., McGuinness, D., Nardi, D., Patel-Schneider, P.F. (eds.): Description Logic Handbook: Theory, Implementation and Applications. Cambridge University Press, Cambridge (2002)
14. Berardi, D., Calvanese, D., De Giacomo, G.: Reasoning on UML class diagrams. Artificial Intelligence 168(1-2), 70–118 (2005)
15. Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Rosati, R.: DL-Lite: Tractable description logics for ontologies. In: Proc. of the 20th Nat. Conf. on Artificial Intelligence (AAAI 2005), pp. 602–607 (2005)
16. Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Rosati, R.: Data complexity of query answering in description logics. In: Proc. of the 10th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR 2006), pp. 260–270 (2006)
17. Calvanese, D., De Giacomo, G., Lenzerini, M.: On the decidability of query containment under constraints. In: Proc. of the 17th ACM SIGACT SIGMOD SIGART Sym. on Principles of Database Systems (PODS 1998), pp. 149–158 (1998)
18. Calvanese, D., Lenzerini, M., Nardi, D.: Description logics for conceptual data modeling. In: Chomicki, J., Saake, G. (eds.) Logics for Databases and Information Systems. Kluwer, Dordrecht (1998)
19. Calvanese, D., Lenzerini, M., Nardi, D.: Unifying class-based representation formalisms. J. of Artificial Intelligence Research 11, 199–240 (1999)
20. Chomicki, J., Toman, D.: Temporal logic in information systems. In: Chomicki, J., Saake, G. (eds.) Logics for Databases and Information Systems, ch. 1. Kluwer, Dordrecht (1998)
21. Combi, C., Degani, S., Jensen, C.S.: Capturing temporal constraints in temporal ER models. In: Li, Q., Spaccapietra, S., Yu, E., Olivé, A. (eds.) ER 2008. LNCS, vol. 5231. Springer, Heidelberg (2008)
22. Etzion, O., Gal, A., Segev, A.: Extended update functionality in temporal databases. In: Etzion, O., Jajodia, S., Sripada, S. (eds.) Dagstuhl Seminar 1997. LNCS, vol. 1399, pp. 56–95. Springer, Heidelberg (1998)
23. Finger, M., McBrien, P.: Temporal conceptual-level databases. In: Gabbay, D., Reynolds, M., Finger, M. (eds.) Temporal Logics – Mathematical Foundations and Computational Aspects, pp. 409–435. Oxford University Press, Oxford (2000)
24. Franconi, E., Sattler, U.: A data warehouse conceptual data model for multidimensional aggregation. In: Proc. of the Workshop on Design and Management of Data Warehouses (DMDW 1999) (1999)

25. Gabbay, D., Kurucz, A., Wolter, F., Zakharyaschev, M.: Many-dimensional modal logics: theory and applications. Studies in Logic. Elsevier, Amsterdam (2003)
26. Gregersen, H., Jensen, J.S.: Conceptual modeling of time-varying information. Technical Report TimeCenter TR-35, Aalborg University, Denmark (1998)
27. Gregersen, H., Jensen, J.S.: Temporal Entity-Relationship models – a survey. IEEE Transactions on Knowledge and Data Engineering 11(3), 464–497 (1999)
28. Gupta, R., Hall, G.: An abstraction mechanism for modeling generation. In: Proc. of ICDE 1992, pp. 650–658 (1992)
29. Hall, G., Gupta, R.: Modeling transition. In: Proc. of ICDE 1991, pp. 540–549 (1991)
30. Hodkinson, I., Wolter, F., Zakharyaschev, M.: Decidable fragments of first-order temporal logics. Annals of Pure and Applied Logic 106, 85–134 (2000)
31. Horrocks, I., Patel-Schneider, P.F., van Harmelen, F.: From SHIQ and RDF to OWL: The making of a web ontology language. Journal of Web Semantics 1(1), 7–26 (2003)
32. Jensen, C.S., Clifford, J., Gadia, S.K., Hayes, P., Jajodia, S., et al.: The Consensus Glossary of Temporal Database Concepts. In: Etzion, O., Jajodia, S., Sripada, S. (eds.) Temporal Databases - Research and Practice, pp. 367–405. Springer, Heidelberg (1998)
33. Jensen, C.S., Snodgrass, R.T.: Temporal data management. IEEE Transactions on Knowledge and Data Engineering 111(1), 36–44 (1999)
34. Jensen, C.S., Soo, M., Snodgrass, R.T.: Unifying temporal data models via a conceptual model. Information Systems 9(7), 513–547 (1994)
35. McBrien, P., Seltveit, A.H., Wangler, B.: An Entity-Relationship model extended to describe historical information. In: Proc. of CISMOD 1992, Bangalore, India, pp. 244–260 (1992)
36. Parent, C., Spaccapietra, S., Zimanyi, E.: The MurMur project: Modeling and querying multi-representation spatio-temporal databases. Information Systems 31(8), 733–769 (2006)
37. Spaccapietra, S., Parent, C., Zimanyi, E.: Modeling time from a conceptual perspective. In: Int. Conf. on Information and Knowledge Management (CIKM 1998) (1998)
38. Spaccapietra, S., Parent, C., Zimanyi, E.: Conceptual Modeling for Traditional and Spatio-Temporal Applications—The MADS Approach. Springer, Heidelberg (2006)
39. Tauzovich, B.: Towards temporal extensions to the entity-relationship model. In: Proc. of the Int. Conf. on Conceptual Modeling (ER 1991). Springer, Heidelberg (1991)
40. Theodoulidis, C., Loucopoulos, P., Wangler, B.: A conceptual modelling formalism for temporal database applications. Information Systems 16(3), 401–416 (1991)

Faceted Lightweight Ontologies

Fausto Giunchiglia[1], Biswanath Dutta[2], and Vincenzo Maltese[1]

[1] Dipartimento di Ingegneria e Scienza dell'Informazione (DISI)
Università di Trento, Trento, Italy
[2] Documentation Research and Training Centre (DRTC), Indian Statistical Institute (ISI),
8th Mile Mysore Road, Bangalore- 560059, India

Abstract. We concentrate on the use of *ontologies* for the categorization of objects, e.g., photos, books, web pages. *Lightweight ontologies* are ontologies with a tree structure where each node is associated a natural language label. *Faceted lightweight ontologies* are lightweight ontologies where the labels of nodes are organized according to certain predefined patterns which capture different aspects of meaning, i.e., *facets*. We introduce facets based on the Analytico-Synthetic approach, a well established methodology from Library Science which has been successfully used for decades for the classification of books. Faceted lightweight ontologies have a well defined structure and, as such, they are easier to create, to share among users, and they also provide more organized input to semantics based applications, such as semantic search and navigation.

Keywords: Ontologies, Lightweight ontologies, facets, classifications, formal classifications.

1 Introduction

Ontologies are being used in different communities, for different purposes and with different modalities. There are various kinds of ontologies, according to the degree of formality, complexity of the graph structure, and expressivity of the language used to describe them [1]. Ontologies have two main applications. They can be used to *describe* objects or they can be used to *categorize* objects. In this paper, we concentrate on the second use, namely, we are interested in the problem of classifying, e.g., photos, Web pages, books.

Lightweight ontologies are ontologies with a tree structure where each node is associated a natural language label. We sometimes speak of *formal lightweight ontologies* meaning ontologies which can be obtained from lightweight ontologies by translating natural language labels into Description Logics (DL) [12] formulas which capture their meaning ([3] provides an example of how such translation can be done). In formal lightweight ontologies, node formulas stand in the subsumption relation, namely a formula in a node is always more general than the formula in the node below [1, 31]. In the following we talk of lightweight ontologies meaning sometimes formal lightweight ontologies. The context always makes clear what we mean.

Lightweight ontologies allow for automated document classification [1, 16], query answering [1, 21] and also for solving the semantic heterogeneity problem among multiple ontologies [15, 18, 19, 20]. They are definitely a very powerful tool which can be exploited towards the automation of reasoning in data and knowledge management. Still, the adoption of (lightweight) ontologies, so far, has not been as widespread as one would have expected when the work on the Semantic Web started. Among the problems which have been identified are the lack of interest or the difficulties on the user side in building such ontologies [4, 5], but also the fact that ontologies developed for one purpose can hardly being reused for other purposes, or by other users [5].

The goal of this paper is to introduce *faceted lightweight ontologies* as a very promising solution to the problem highlighted above. Faceted lightweight ontologies are defined in terms of *facets*. Recently, facets have been adopted with great success for the design of interfaces to web sites. See, for instance the survey by La Barre [23] and in particular the work done in Flamenco[1] (see for instance [24]), but see also [7,8,9] as an application to knowledge management which is somewhat related, in spirit, to our work. We construct facets following the approach which was first devised by Ranganathan at the beginning of the last century [22] [2] and, in particular, the POPSI Methodology, originally introduced in [26].

Taking the terminology of Library Science, facets are "*aspects of meaning*". They formalize, for any given domain (e.g., medicine, sports, music, science), the main characteristics of that domain and, in particular, the entities or objects which belong to that domain (e.g. in medicine, the body parts), the properties of objects (e.g., in medicine, the various kinds of disease) and the actions which can be taken (e.g., in medicine, surgery or medication). More precisely, a facet is a hierarchy of homogeneous group of terms (nodes), where each term in the hierarchy denotes a primitive atomic concept. Thus we have hierarchies of entities, properties, actions, and so on. We call *background knowledge* [17,14], a *faceted representation scheme,* namely a set of facets that represent the system a-priori knowledge about the domains of interest (see also [13] for an early attempt of defining a faceted representation schema not based on Ranganathan's theory). A faceted representation scheme allows for post-coordination, namely, for constructing complex labels (in Library Science terminology, also called *subjects*) by combining terms from facets at both indexing, classification and searching time. Faceted lightweight ontologies are lightweight ontologies where node labels (formulas) contain only atomic concepts which correspond to primitive concepts taken from the background knowledge.

The rest of the paper is organized as follows. Section 2 introduces and formally defines (classification) lightweight ontologies. Section 3 introduces facets. Section 4 introduces faceted subjects and, then, the notion of faceted lightweight ontology. Finally, Section 5 shows, via an example, how a faceted subject can be constructed according to the POPSI subject indexing system. Section 6 concludes the paper.

[1] http://flamenco.berkeley.edu

[2] This theory is widely recognized as a fundamental methodology that guides in the organization of the knowledge in a given domain (see for instance [30]) in terms of basic subjects and relations between them.

2 Lightweight Classification Ontologies

Ontologies have been used for centuries in different communities, for different purposes and with different modalities. The concept originated more than two thousand years ago from philosophy and more specifically from Aristotle's theory of categories[3]. The original purpose was to provide a categorization of all existing things in the world. Ontologies have been lately adopted in several other fields, such as Library and Information Science (LIS), Artificial Intelligence (AI), and more recently in Computer Science (CS), as the main means for describing how classes of objects are correlated, or for categorizing what archivists generically call documents.

Many definitions of ontologies have been provided. According to the most quoted, an ontology is "*an explicit specification of a conceptualization*" [10]. Their main purpose is to favour interoperability by providing a common terminology and understanding of a given domain of interest, which in turn allows for the assignment of clear meanings to information items. There are however different kinds of ontologies, or, in other words, several more specific concepts, according to the degree of formality and expressivity of the language used to describe them (see [2] for a discussion). They range from informal representations like user classifications (e.g. the structure of folders in a file system) and web directories (e.g. DMOZ, Yahoo! and Google[4]), to progressively more formal representations like enumerative classification schemes (e.g. the Dewey Decimal Classification[5] (DDC) and the Library of Congress Classification[6] (LCC)), Knowledge Organization Systems (KOS) such as thesauri (e.g. AGROVOC, NALT, AOD, and HBS) and faceted classification schemes (e.g., the Colon Classification), and, ultimately, formal ontologies which are expressed into a logic formal language and represented using formal specifications such as DL or OWL.

For the purpose of this work, however, following the terminology provided in [1], the core distinction is between

1. ontologies which are mainly used to *describe* objects, also called *descriptive ontologies*, and
2. ontologies which are mainly used to *categorize* objects, also called *classification ontologies*.

This distinction is reflected into the underlying semantics taken as reference, namely the *real world semantics* and the *classification semantics* described below. Based on this distinction then we further refine the notion of classification ontology into the notion of *classification lightweight ontology*, which is actually the core notion needed in this paper. Let us analyze these notions in detail.

2.1 Descriptive Ontologies

In descriptive ontologies, *concepts represent real world entities, e.g., the extension of the concept animal is the set of real world animals*, which can be connected via

[3] http://plato.stanford.edu/entries/aristotle-categories/
[4] http://dmoz.org/; http://dir.yahoo.com/; http://directory.google.com/
[5] http://www.oclc.org/dewey/
[6] http://www.loc.gov

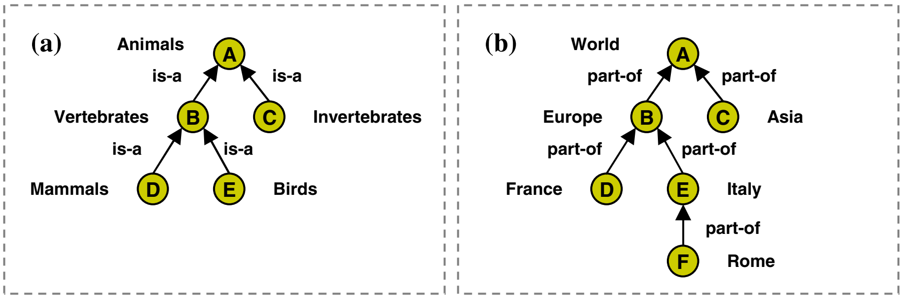


Fig. 1. (a) An is-a ontology; (b) A part-of ontology

relations of the proper kind. The purpose of descriptive ontologies is to specify the terms used in their original meaning, according to the nature and the structure of the domain they model [11]. Two typical relations are used to construct the trees (also called the taxonomies) which provide the backbone to these ontologies and they are *is-a* (Genus-species) and *part-of* (Whole-part) relations. In Fig. 1 (a), (b) we provide two examples of descriptive ontologies, based on these two relations, where each node represents a concept and each arrow represents a relation between them. The direction of the arrows represents the direction of the relations. Mixed situations are also possible.

It is worth noticing that, when translating these ontologies in DL, the *is-a* relation is translated into subsumption ($\sqsubseteq$) or, more precisely, it is assumed to imply subsumption, while this is not the case for *part-of*. Therefore *is-a* constitutes the basic backbone of the subsumption based hierarchical structure of a domain.

2.2 Classification Ontologies

Ontologies in classification semantics are built with the goal of indexing documents. As a consequence, *the extension of each concept (label of a node) is the set of documents about the entities or individual objects described by the label of the concept* [1,2]. For example, the extension of the concept animal is “the set of documents about animals” of any kind. This has three main consequences.

The first is that the semantic relation holding between nodes which are one above the other is *always* the *subset* relation. In other words the set of documents which can be classified in a node is always a subset of the documents which can be classified in the node above (and this motivates some techniques for minimizing the number of nodes where a document is classified, for instance the *get-specific* principle, see [16] for a formalization of this principle and its use in automatic classification). Fig. 2 (a), (b) provide the classification semantics version of the two ontologies reported in Fig. 1 (a), (b). As it can be noticed from Fig. 2, the standard relations of descriptive ontologies are translated into relations among sets. Thus, *is-a*, but also to *part-of*, when transitive, and *instance-of*, are translated into *subset*, while the others correspond to *overlap* ($\sqcap$). Fig. 2 (b) provides a case where the *part-of* relations of Fig. 1 (b) are translated into *subset* in classification semantics. One example where this is not possible is the chain of relations: handle *part-of* door *part-of* school *part-of* school system.

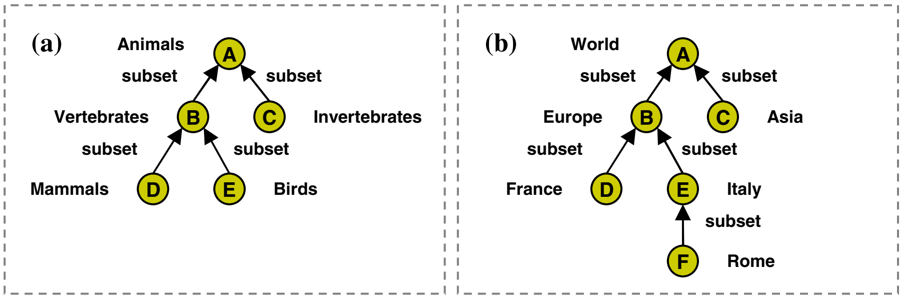


Fig. 2. Two ontologies in classification semantics

The second consequence is that, in the DL translation of classification ontologies, the subset relation is translated into subsumption between the formulas of nodes which are one above the other. It is important to observe that the DL translation of the same ontology, if taken with real world semantics or with classification semantics, leads to a different DL theory (compare again Fig. 1(b) and Fig. 2(b))).

Notice that the labels in both ontologies in Fig. 2 are such that each of them represents a proper subset of the label of the node above. Thus, for instance, *vertebrates* represents a proper subset of *animals*. However, and this is the third consequence, in classification ontologies, the situation above can be generalized to consider labels which denote sets which are not in the subset relation, but, rather, in the *overlap* relation. As a matter of fact, this is what happens in most classification ontologies [2]. Consider for instance the classification ontology in Fig. 3 (a). The intuition is that node B should contain all documents which are about "*the research on Java*". In other words, the meaning of a node (so-called "*concept at a node*" in [1,14]) can be constructed by taking the DL conjunction of (semantically, the intersection of the sets denoted by) the concepts of all the labels in the path from the root to the node itself. The application of this rule to the example in Fig. 3 (a) leads to the ontology in Fig. 3(b). As it can be noticed, the concept associated to a node is in the subsumption relation with any node above and this is obtained by applying the conjunction operator over the path. The numbers after each label are used to denote the concept which

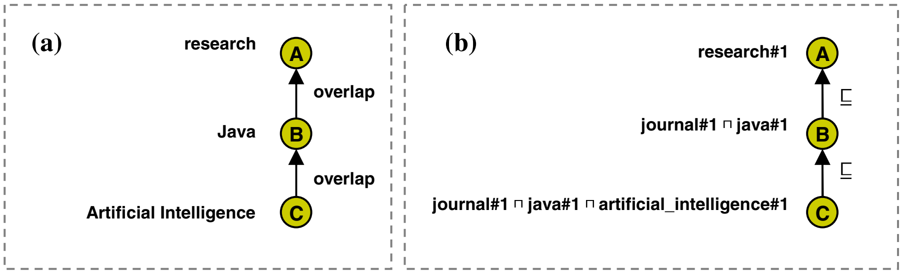


Fig. 3. (a) A classification ontology with no *subset-of* relation between labels, (b) the corresponding formal ontology

is obtained by disambiguating it (each word may correspond to more than one concept, e.g. Java can be an island, a programming language or a kind of coffee beans). It is easy to notice how the situation in Fig. 3 (a) collapses into the situations in Fig. 2 (a) once we return to the subset relation: all the conjunctions become redundant due to the fact that if $A \sqsubseteq B$, then $A \sqcap B$ is equivalent to A.

2.3 Lightweight Classification Ontologies

All the theory on classification of Library Science and, as a consequence, the theory of facets, as originally devised by Ranganathan and later refined in the POPSI methodology, is based on classification semantics. And it is correctly so, as these methodologies were invented in order to classify books and position them in shelves. In the following of this paper we also concentrate on classification ontologies and classification semantics. The motivation is quite similar to that in Library Science. It is a fact that, e.g., on line catalogs, file systems, web directories and library classifications are used for classifying objects and can be translated, exactly or with a certain degree of approximation, into classification ontologies.

As a matter of fact, in all applications from Library Science and also in our reference applications, the classification ontologies which are needed are quite simple and consists of trees, possibly multi-rooted, where most of the nodes in the father-child relation do *not* have labels whose denotation stands in the subset relation. Each node label can therefore be translated into a logic formula (typically built as a combination of conjunctions and disjunctions of atomic concepts) representing the meaning of the node taking into account its context, namely the path from the root to the node [3]. This leads to the definition of classification lightweight ontology, as originally defined in [25] (the word "classification" did not appear in the original definition):

A ***lightweight classification ontology*** *O is a rooted tree* $<N,E,L^F>$ *where:*

a) N is a finite set of nodes;

b) E is a set of edges on N;

c) L^F *is a finite set of labels expressed in a Propositional DL language such that for any node* $n_i \in N$*, there is one and only one label* $l_i^F \in L^F$*;*

d) $l_{i+1}^F \sqsubseteq l_i^F$ *with* n_i *being the parent of* n_{i+1}*.*

3 Facets

According to the Analytico-Synthetic approach [22,27], facets are defined following two steps:

1. examine the field (domain) to identify relevant terms. They can be gained by consulting domain experts and all sorts of information sources over the domain. This process starts in the so called "*idea plane*", the language independent conceptual level, where primitive concepts are identified. Each identified concept, in turn, is expressed in the "*verbal plane*" in a given language, for example in English, trying to articulate the idea *coextensively*, namely identifying a term which exactly and unambiguously expresses the concept;

2. group the identified terms (also called *isolate ideas*) according to their common properties or characteristics, and order them (in hierarchies) in a meaningful sequence. The set of homogeneous terms form a *facet*. For example, *Nose, Larynx, Trachea, Bronchi, Lung, Pleural sac, Mediasinum* form a facet called *Respiratory system* (these entities are in the *part-of* relation with *Respiratory system*). Now the terms *Outer nose* and *Nasal,* which are again *part-of* Nose, can form a facet called *Nose* which will be treated as *sub-facet* of the facet *Respiratory system.*

These two steps construct a *faceted representation scheme* and correspond to what in our previous work we call the definition and construction of the so called *background knowledge* [17, 21], namely the a-priori knowledge which must exist in order to make semantics effective. Notice that the grouping of terms of step 2 have real world semantics, namely, they are descriptive ontologies which are formed using *part-of*, *is-a* and *instance-of*. Facets have the following two key properties:

1. They are organized as a set of independent *domains* which are completely modular and can be developed independently.
2. For each domain, facets are grouped into specific elementary *categories*. Originally, Ranganathan defined five fundamental categories: *Personality*, *Matter*, *Energy*, *Space* and *Time (PMEST)*. Later on, Bhattacharyya proposed a refinement which consists of four main categories, called DEPA: *Discipline* (D) (what we now call a domain), *Entity* (E), *Property* (P) and *Action* (A), plus another special category, called Modifier (m).

In our approach we organize facets according to the DEPA categories. Let us describe them in some detail:

– **Discipline** (or **domain**): it includes conventional fields of study (e.g., *Library Science*, *Mathematics* and *Physics*), applications of the traditional pure disciplines (e.g., *Engineering* and *Agriculture*), any aggregates of such fields (e.g., *Physical Sciences* and *Social sciences*), or also, in more modern terms, fields like *music*, *sports*, *computer science*, and so on.
– **Entity:** the elementary category Entity is manifested in perceptual correlates or in conceptual existence. It is distinct from their properties and actions performed by them or on them. Basically the concepts represent the core idea of a domain treated as under this elementary category. For example, *"Teachers"*, *"Students"*, *"Courses"* are the core concepts to a domain *"Education"*.
– **Property:** it includes concepts denoting quantitative or qualitative attributes. For example, *Quality*, *Quantity*, *Measure*, *Weight*, *Taste*, etc;
– **Action:** it includes concepts denoting the notion of "*doing*". It includes "*processes*" and "*steps*" of doing. An action can manifest as "*Self-action*" or "*External action*". A self-action is an action done by some agent (explicit or implicit) on or in itself. For example, *Imagination*, *Interaction*, *Reaction*, *Reasoning*, *Thinking*, etc. An external action is an action done by some agent (explicit or implicit) on a concept of any of the elementary categories described above. For example, *Organization*, *Cooperation*, *Classification*, *Cataloguing*, *Calculation*, *Design*, etc.

– **Modifier:** it includes concepts used or intended to be used to qualify other concepts. With the help of a modifier, the extension of a concept is decreased and the intension is increased without disturbing its conceptual wholeness. For example, “Mining in India”, here India modifies Mining. By implication, any concept from the elementary categories above or combination of two or more concepts may serve as the basis of deriving a modifier. There are many kinds of modifiers, in particular we can distinguish *common modifiers* (e.g., *space-modifier*, *time-modifier*, *environment-modifier*, *form-modifier*, *language modifier*) and *special modifiers* (e.g., *Infectious*, *Bacterial*, *Fungus*, etc. modify the concept “*Diseases*” in the *Medicine* domain). Common modifiers are common to all disciplines used to modify manifestations of more than one elementary category, occurring singly or in combination. Special modifiers modify manifestations of one and only one elementary category. However, following the principle of reusability (described below), some modifiers can be shared by a set (but not all) domains (for instance chemical substances are used both in Chemistry and in Agriculture, possibly under different categories).

The basic rule for formulating subject headings is *Discipline* (base) first, followed by *Entity* (core), which is followed by *Property* and/or *Action*. Property and/or Action may be further followed by Property and/or Action as the case may be, followed by *Common modifiers*. The *species/types* and/or *modifiers* and/or *parts* and/or *constituents* for each of the *elementary categories* follow immediately the manifestation to which they are respectively *species/types* or *modifiers* or *parts* or *constituents.* In Fig. 4 we provide an example of facets grouped in the DEPA categories in the Medicine domain. Notice that, even if this is not the case in the example above, in each category we can potentially have more than one facet.

Facets possess some essential properties as listed below:

– **Hospitability:** they are easily extensible. New terms representing new knowledge can be accommodated without difficulty in the hierarchical structure. Terms in the hierarchies are clearly defined, mutually exclusive and collectively exhaustive.
– **Compactness:** facet based systems need less space with respect to the other hierarchical knowledge organization systems to classify the universe of knowledge. There is no explosion of the possible combinations as the basic elements (facets) are taken in isolation.
– **Flexibility:** hierarchical knowledge organization systems are mostly rigid in their structure, whereas facet based systems are flexible in nature.
– **Reusability:** a facet based ontology developed for a particular domain could be partially usable into another related domain.
– **Clear, but rigorous, structure:** the faceted approach aims at the identification of the logical relations between concepts and concepts groups. Sibling concepts must share a common characteristic.
– **The methodology:** a strong methodology for the analysis and categorization of concepts along with the existence of reliable rules for synthesis is provided.
– **Homogeneity:** a facet represents a homogeneous group of concepts, according to the specified common characteristic(s).

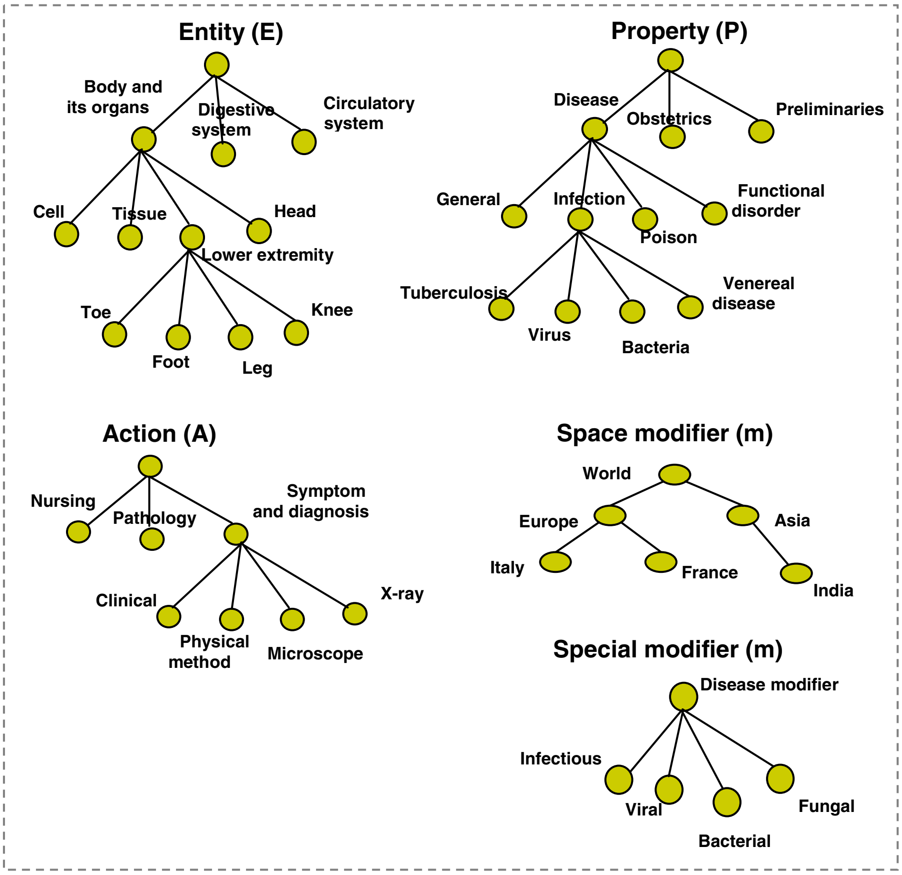


Fig. 4. The set of facets for the *Medicine* domain

4 Faceted Lightweight Ontologies

Once the background knowledge is constructed, the next step is to see how to use facets in order to index or classify documents (in our case, inside lightweight ontologies). As from above, for us this corresponds to associating to each document and node in a classification a DL formula [1,2]. This step happens inside what Ranganathan called the "*notational plane*". Here an unambiguous notation is used to synthetically attach meaning and provide order to the managed objects, typically books on the shelves. Following G. Bhattacharyya [26], the key idea is to associate to a node or document a *subject*, namely "*a piece of non-discursive information that summarises indicatively what a book or document (any body of information) is about*". A subject, in our terms, is the label and corresponding concept associated to a document or a node in a lightweight ontology. Since in lightweight ontologies we use classification semantics, a document will be classified in any node whose subject is more general than the subject of the document [1,16].

We define subjects in terms of facets. The key intuitions are three:

1. We associate to each term in the subject a label and corresponding concept taken from a faceted classification scheme (in POPSI the concept is given by the preferred term and its context);
2. For each term in a facet, the context is constructed by associating to it all the terms from the root to the term itself, thus disambiguating the intended concept. Notice that this means that, in the step from the background knowledge to the subject concept, we need to translate from real world semantics (used in the background knowledge) to classification semantics (used in lightweight ontologies).
3. Each subject contains terms (concepts) from *potentially all* the DEPA categories, thus allowing for the complete disambiguation of the subject. However, the user is supposed to provide, explicitly or implicitly, at least the *discipline* and the main *entity*.

In POPSI, in order to construct the context, each *leading heading* (also called *lead term* or *term-of approach*) is followed by the *context heading*, namely the set of auxiliary terms which preserves the context (in terms of the *discipline* and the path from the root of the facet to the term). For instance, the context of the term *Cell* is:

Cell (lead term)
Medicine, Body and its organs > Cell (context heading)

In the above example, "," separates the *isolate ideas* (i.e. the concepts) belonging to the different fundamental categories as shown in section 3, while ">" identifies the increasing intension and decreasing extension of isolate ideas within a facet. Notice that, from Fig. 4 above, Medicine is the name of the domain while the second part is the complete path in the entity facet. Consider, furthermore, the subject "*Microscopic diagnosis of bacterial viruses on cells in India*". Its terms are completely contextualized in POPSI as follows (the sequence of concrete steps necessary to identify them is described in the next section):

(Domain):	Medicine,
(Entity):	Body and its organs > Cell,
(Property):	Disease > Infection > Virus,
(Modifier of P.)	Bacterial,
(Action):	Symptom and diagnosis > Microscope,
(Space modifier):	Asia > India

The main advantage of the faceted approach is that it makes explicit the logical relations among the concepts and concept groups and removes the limitations of traditional hierarchies. It allows for viewing a complex entity from a variety of perspectives or from different angles. For example, a *cow* can be described as an animal, as a pet, as a food item, as a commodity, as a God for a particular community, and so on, depending on the domain. Therefore, each time, by providing the context, the faceted approach allows for the representation of different concepts.

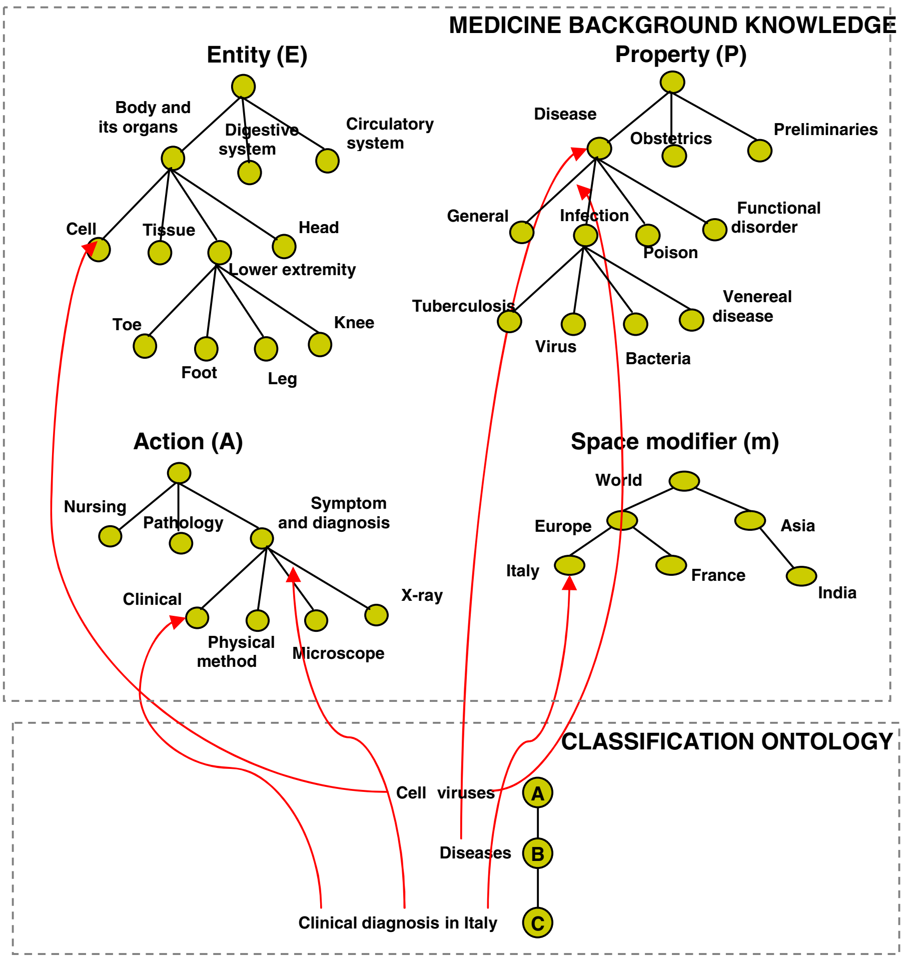


Fig. 5. A faceted lightweight ontology

Based on the notion of subject, we can now define *a faceted lightweight (classification) ontology* as follows:

A ***faceted lightweight (classification) ontology*** *is a lightweight ontology where each term and corresponding concept occurring in its node labels must correspond to a term and corresponding concept in the background knowledge, modeled as a faceted classification scheme.*

Fig. 5 provides an example of how this can be done. Notice that in faceted lightweight ontologies there might be nodes, as in Fig. 5, whose labels contain terms from multiple DEPA categories, while in other cases we will have one node per DEPA category. The more terms and corresponding DEPA categories there will be, the more specific the lightweight classification ontology will be.

5 Subject Indexing

How do we use faceted classifications schemes, in practice? As already mentioned in the previous section, documents will be classified under those nodes whose subject is more general than theirs. But, the real challenge is that in most cases the subject specification is only partial. To this extent, POPSI provides a methodology for providing the missing *contextual information*. The solution lies mainly in the appropriate representation of the extension and intension of the thought content (subject matter) of the indexed documents. Let us now discuss the steps involved in POPSI in deriving the subject strings starting from the titles associated to documents to index along with an example. Let us consider the example of a subject given in the previous section:

"*Microscopic diagnosis of bacterial viruses on cells in India*".

The analysis is organized in eight steps, as described below:

Step 1 (Analysis of the subject indicative expression): it concerns the analysis of the subject indicative expression pertaining to the source of information. It may be the title of a book, article etc. For the example above, we derive the following terms:

D = Medicine (implicit in the above title)
E = Cells (explicit)
P = Viruses (explicit)
m of P = Bacterial (explicit)
A = Microscopic diagnosis (explicit)
m = India (explicit) (Space modifier)

In our approach this step is performed analogously. Notice that implicit categories must be provided manually by the user or computed automatically by the system.

Step 2 (Formalization of the Subject Proposition): in this stage the formalization of the sequence of the terms appearing in the subject derived by Step 1 (Analysis) is done. According to the principles of sequence, the components are sequenced in the following way:

Medicine (D), Cells (E), Viruses (P), Bacterial (m of P), Microscopic diagnosis (A), India (m)

In our approach this step is not needed.

Step 3 (Standardization of the Subject Proposition): It consists in the identification of the standard terms, when synonyms of the same term are available, denoting the atomic concepts present in the subject proposition. For our example, this step is not applicable. So, the subject proposition remains the same:

Medicine, Cells, Viruses, Bacterial, Microscopic diagnosis, India

In our approach this step is performed analogously. This information is codified in the background knowledge.

Step 4 (**Modulation of the Subject Proposition**)**:** It consists of augmenting the standardized subject proposition by interpolating and extrapolating, as the case may be, the successive super-ordinates of each concept by using standard terms with indication of their synonyms. In practice, it corresponds to the identification of corresponding contextual terms, namely the correct disambiguation of each concept used, providing the right amount of hierarchically related concepts:

> Medicine, Body and its organs > Cell, Disease > Infection > Virus, Bacterial, Symptom and diagnosis > Microscope, Asia > India

In our approach this step is performed analogously: we extract from the background knowledge, the concept of each natural language term occurring in the subject.

Step 5 (**Preparation of the Entry for Organizing the Classification**)**:** This step consists of preparing the main entries in the so called associative index in alphabetical arrangement. This is done by assigning a systematic set of numbers as given in [26] to indicate the categories and positions of the subject propositions. In our example:

> Medicine 8 Body and its organs 8.3 Cell 8.2 Disease 8.2.4 Infection 8.2.4.4 Virus 8.2.4.4.6 Bacterial 8.2.1 Symptom and diagnosis 8.2.1.4 Microscope 4 Asia 4.4 India

In our approach this step is not needed.

Step 6 (**Decision about the Terms-of Approach**)**:** It consists of deciding the terms-of approach, namely the lead terms, for generating associative classifications, and of controlling synonyms. For controlling synonyms, each standard term is to be referred to from each of its synonyms. For example (this is not part of our running example),

> Chemical treatment (Medicine)
> *see*
> Chemotherapy

In our approach this step is not needed.

Step 7 (Preparation of the Entries for Associative Classification): It consists of preparing entries under each term-of-approach by cyclic permutation. For example (all other entries can be treated similarly):

Body and its organ
> Medicine, Body and its organs > Cell, Disease > Infection > Virus, Bacterial, Symptom and diagnosis > Microscope, Asia > India

Cell
> Medicine, Body and its organs > Cell, Disease > Infection > Virus, Bacterial, Symptom and diagnosis > Microscope, Asia > India

In our approach this step is not needed.

Step 8: Alphabetical Arrangement of Entries
It consists of arranging all the entries including the reference entries in alphabetical sequence according to a set of standardized rules ignoring the signs and punctuation marks.

```
Asia
  Medicine, Body and its organs > Cell, Disease > Infection > Virus, Bacterial,
   Symptom and diagnosis > Microscope, Asia > India
Bacterial
    Medicine    …        India
 …
   …
Virus
   Medicine     …        India
```

In our approach this corresponds to indexing or classifying inside a faceted lightweight ontology using the concepts of the nodes and the documents.

6 Conclusion

In this paper, we have introduced the notion of faceted lightweight ontology as a lightweight ontology whose terms are extracted from a background knowledge organized in terms of facets. Using facets allows us to have much more control on the language and concepts used to build ontologies and also on their organization, which in general will exploit the structure and terms of the four basic DEPA categories.

This work has been done as part of the FP7 Living Knowledge FET IP European Project.

References

1. Giunchiglia, F., Marchese, M., Zaihrayeu, I.: Encoding Classifications into Lightweight On-tologies. Journal of Data Semantics 8, 57–81 (2006); Short version in: Proceedings of the 3rd European Semantic Web Conference (ESWC) (2006)
2. Giunchiglia, F., Zaihrayeu, I.: Lightweight ontologies. In: S. (ed.) Encyclopedia of Database Systems (2008)
3. Zaihrayeu, I., Sun, L., Giunchiglia, F., Pan, W., Ju, Q., Chi, M., Huang, X.: From web directories to ontologies: Natural language processing challenges. In: Aberer, K., Choi, K.-S., Noy, N., Allemang, D., Lee, K.-I., Nixon, L., Golbeck, J., Mika, P., Maynard, D., Mizoguchi, R., Schreiber, G., Cudré-Mauroux, P. (eds.) ASWC 2007 and ISWC 2007. LNCS, vol. 4825, pp. 623–636. Springer, Heidelberg (2007)
4. Mai, J.-E.: Classification in Context: Relativity, Reality, and Representation. Knowledge Organization 31(1), 39–48 (2004)
5. Duval, E., Hodgins, W., Sutton, S., Weibel, S.L.: Metadata Principles and Practicalities. DLib Magazine 8(4) (2002), `http://www.dlib.org/dlib/april02/weibel/04weibel.html`

6. Nicholson, D., Neill, S., Currier, S., Will, L., Gilchrist, A., Russell, R., Day, M.: HILT: High Level Thesaurus Project – Final Report to RSLP & JISC. Centre for Digital Library Research, Glasgow, UK (2001), `http://hilt.cdlr.strath.ac.uk/Reports/Documents/HILTfinalreport.doc`
7. Tzitzikas, Y., Armenatzoglou, N., Papadakos, P.: FleXplorer: A Framework for Providing Faceted and Dynamic Taxonomy-Based Information Exploration. In: DEXA Workshops, pp. 392–396 (2008)
8. Tzitzikas, Y., Analyti, A., Spyratos, N., Constantopoulos, P.: An algebra for specifying valid compound terms in faceted taxonomies. Data Knowl. Eng. (DKE) 62(1), 1–40 (2007)
9. Tzitzikas, Y., Spyratos, N., Constantopoulos, P., Analyti, A.: Extended Faceted Ontologies. In: Pidduck, A.B., Mylopoulos, J., Woo, C.C., Ozsu, M.T. (eds.) CAiSE 2002. LNCS, vol. 2348, pp. 778–781. Springer, Heidelberg (2002)
10. Gruber, T.R.: A translation approach to portable ontology specifications. Knowledge Aquisition 5(2), 199–220 (1993)
11. Guarino, N.: Helping people (and machines) understanding each other: The role of formal ontology. In: Meersman, R., Tari, Z. (eds.): CoopIS/DOA/ODBASE 2004. LNCS, vol. 3290, p. 599. Springer, Heidelberg (2004)
12. Baader, F., Calvanese, D., McGuinness, D., Nardi, D., Patel-Schneider, P.F.: The Description Logic Handbook: Theory, Implementation and Applications. Cambridge University Press, Cambridge (2002)
13. Soergel, D.: A Universal Source Thesaurus as a Classification Generator. Journal of the American Society for Information Science 23(5), 299–305 (1972)
14. Giunchiglia, F., Yatskevich, M., Shvaiko, P.: Semantic Matching: algorithms and implementation. Journal on Data Semantics IX (2007)
15. Giunchiglia, F., McNeill, F., Yatskevich, M., Pane, J., Besana, P., Shvaiko, P.: Approximate Structure-Preserving Semantic Matching. In: 7th International Conference on Ontologies, Databases and Applications of Semantics (ODBASE 2008), Monterrey, Mexico (November 2008)
16. Giunchiglia, F., Zaihrayeu, I., Kharkevich, U.: Formalizing the get-specific document classification algorithm. In: Kovács, L., Fuhr, N., Meghini, C. (eds.) ECDL 2007. LNCS, vol. 4675, pp. 26–37. Springer, Heidelberg (2007)
17. Giunchiglia, F., Shvaiko, P., Yatskevich, M.: Discovering Missing Background Knowledge in Ontology Matching. In: Brewka, et al. (eds.) Proceedings: 17th European Conference on Artificial Intelligance - ECAI 2006, Riva del Garda, Italy, August 29- September 1, 2006, vol. 141, pp. 382–386. Leipzig University, Germany (2006)
18. Giunchiglia, F., Shvaiko, P., Yatskevich, M.: Semantic schema matching. In: Meersman, R., Tari, Z. (eds.) OTM 2005. LNCS, vol. 3760, pp. 347–365. Springer, Heidelberg (2005)
19. Giunchiglia, F., Yatskevich, M., Giunchiglia, E.: Efficient semantic matching. In: Gómez-Pérez, A., Euzenat, J. (eds.) ESWC 2005. LNCS, vol. 3532, pp. 272–289. Springer, Heidelberg (2005)
20. Giunchiglia, F., Yatskevich, M.: Element Level Semantic Matching. In: Workshop on Meaning Coordination and Negotiation. ISWC 2004, Hiroshima, Japan (November 2004)
21. Giunchiglia, F., Kharkevich, U., Zaihrayeu, I.: Concept Search: Semantics Enabled Syntactic Search. In: Semantic Search 2008 workshop (SemSearch 2008) at the 5th European Semantic Web Conference (ESWC 2008) (2008)
22. Ranganathan, S.R.: The Colon Classification. In: Artandi, S. (ed.) The Rutgers Series on Systems for the Intellectual Organization of Information, vol. IV. Graduate School of Library Science, Rutgers University, New Brunswick (1965)
23. La Barre, K.: Adventures in faceted classification: A brave new world or a world of confusion? In: Knowledge organization and the global information society: proceedings 8th ISKO conference, London, July 13-16 (2004)

24. Hearst, M.: Design Recommendations for Hierarchical Faceted Search Interfaces. In: ACM SIGIR Workshop on Faceted Search, Seattle, WA (2006)
25. Giunchiglia, F., Maltese, V., Autayeu, A.: Computing minimal mappings. University of Trento, DISI Technical Report (2008)
26. Bhattacharyya, G.: POPSI: its fundamentals and procedure based on a general theory of subject indexing languages. Library Science with a Slant to Documentation 16(1), 1–34 (1979)
27. Ranganathan, S.R.: Prolegomena to library classification. Asia Publishing House, London (1967)
28. Ranganathan, S.R.: Elements of library classification, p. 3. Asia Publishing House, Bombay (1960)
29. Aptagiri, D.V., Gopinath, M.A., Prasad, A.R.D.: A frame-based knowledge representation paradigm for automating POPSI (1995)
30. Broughton, V.: The need for a faceted classification as the basis of all methods of information retrieval. Aslib Proceedings 58(1/2), 49–72 (2006)
31. Zaihrayeu, I., Marchese, M., Giunchiglia, F.: Encoding Classifications into Lightweight Ontologies. In: Sure, Y., Domingue, J. (eds.) ESWC 2006. LNCS, vol. 4011, pp. 80–94. Springer, Heidelberg (2006)

The Ontological Level: Revisiting 30 Years of Knowledge Representation

Nicola Guarino

ISTC-CNR, Laboratory for Applied Ontology, Via alla Cascata 56/C, Trento, Italy
nicola.guarino@cnr.it

Abstract. I revisit here the motivations and the main proposal of paper I published at the 1994 Wittgenstein Symposium, entitled "The Ontological Level", in the light of the main results achieved in the latest 30 years of Knowledge Representation, since the well known "What's in a link?" paper by Bill Woods. I will argue that, despite the explosion of ontologies, many problems are still there, since there is no general agreement about having ontological distinctions built in the representation language, so that assumptions concerning the basic constructs of representation languages remain implicit in the mind of the knowledge engineer, and difficult to express and to share. I will recap the recent results concerning formal ontological distinctions among unary and binary relations, sketching a basic ontology of meta-level categories representation languages should be aware of, and I will discuss the role of such distinctions in the current practice of knowledge engineering.

Keywords: ontology, knowledge representation, identity, rigidity, OntoClean.

1 Introduction

About 25 years ago, Ron Brachman, Richard Fikes and Hector Levesque [5] published a seminal paper describing a hybrid knowledge representation system (KRYPTON) built around two separate components reflecting the natural distinction between terms and sentences: the TBox (for *terminological knowledge*) and the ABox (for *assertional knowledge*). Terms were represented in the TBox by a structured formalism that was an ancestor of modern description logics, allowing the knowledge engineer to form composite descriptions corresponding to noun phrases like "an igneous rock", "a grey rock", or "a family with no children". A terminological knowledge base can be seen as a network of *analytic relationships* between such descriptions. If the basic vocabulary and the description-forming rules are rich enough, such a network can easily become quite complicated, due to the possibility of forming complex descriptions. For instance, even with a small set of attributes denoting different properties of rocks, it is easy to come up with a relatively complex taxonomy, as the authors point out while presenting Fig. 1.

In this context, the authors discussed the effects of a query such as "How many rock kinds are there?". They observed that, despite its commonsense simplicity, this is a "dangerous question to ask", as it cannot be answered by simply looking at the nodes subsumed by 'rock' in the network, since the language allows them to

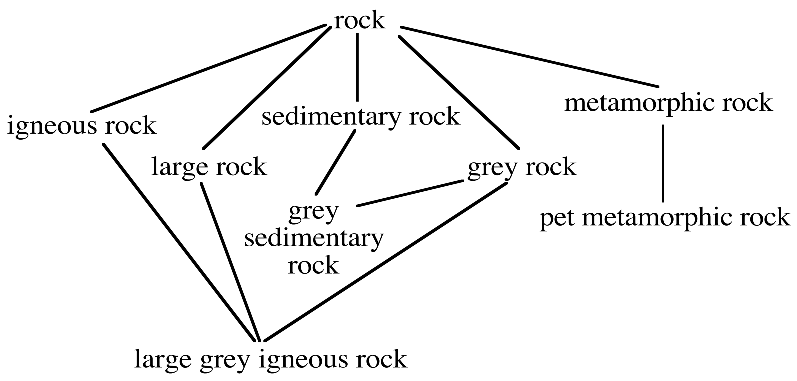


Fig. 1. Kinds of rocks (From [5])

proliferate easily, as soon as new attributes are added to the vocabulary. Hence they proposed a functional approach to knowledge representation designed to only answer "safe" queries that are about analytical relationships between terms, and whose answers are independent of the actual structure of the knowledge base, like "a large grey igneous rock is a grey rock".

It is clear that, in this example, Brachman and colleagues understood the term "rock kind" in a very simple, minimalist way (perhaps as synonymous with "rock class"), ignoring the fact that, for many people, there are just three kinds of rocks, as taught at high school: Igneous, Metamorphic, and Sedimentary. On the other hand, two of the same authors, in an earlier paper on terminological competence in knowledge representation [6] stressed the importance of distinguishing an "enhancement mode transistor" (which is "a kind of transistor") from a "pass transistor" (which is "a role a transistor plays in a larger circuit").

So why was this distinction ignored? My own conclusion is that important issues related to the different ontological assumptions underlying our use of terms have been simply given up while striving for logical simplification and computational tractability. As a consequence, most representation languages, including "ontology languages" like OWL, do not offer constructs able to distinguish among terms having similar logical structure but different ontological implications. In our example, clearly "large rock" and "sedimentary rock" have the same logical structure, being both interpreted as the conjunction of two (primitive) logical properties; yet we tend to believe that there is something radically different between the two: why? To answer this question we have to investigate:

- the nature of the primitive properties "being a rock", "being large", and "being sedimentary";
- the way they combine together in a structured term, while modifying each other.

Unfortunately, while current representation languages offer us powerful tools to build structured descriptions whose formal semantics is carefully controlled to provide efficient reasoning services, still no agreement has been reached concerning the need to adopt proper mechanisms to control the ontological commitments of structured representation formalisms, as their semantics is completely neutral with respect to the nature of the primitive components and the structuring relationships.

To see another instance of this unfortunate situation, involving binary relations instead of unary properties as in the previous case, consider the old example brought about by Bill Woods' in its classic "What's in a Link?" paper [38]:

```
JOHN
    HEIGHT: 6 FEET
    HIT: MARY
```

As Woods observed, in this case the two relations 'Height' and 'Hit' have certainly a different ontological nature, but nothing excludes, in the semantics of description logics or similar structured representation formalisms, them from being considered as "attributes" or "roles" (in the description logic's sense), since these constructs are understood as *arbitrary* binary relations. So, more than 30 years later, Woods' problem cannot be considered as solved.

Indeed, ontologies have exploded nowadays, but many problems are still there: we have now ontology languages, but despite a fair amount of results concerning the formal analysis of ontological distinctions like the ones mentioned before – including OntoClean [20, 21] and the related work on the ontological characterization of unary properties [18, 19, 31], as well as extensive analyses of fundamental binary relations such as parthood, location or dependence [32, 37, 2, 9, 33, 34, 12] – there is still no general agreement about having such distinctions built in the language, so that assumptions such as those concerning the basic constructs of representation languages remain implicit in the mind of the knowledge engineer, however difficult to express and share. A concrete proposal in this direction has been made in [23], where an ontologically well-founded profile for UML is proposed, which constrains the semantics of UML modeling elements in the light of ontological distinctions mainly inspired to OntoClean. This is still a preliminary work, however, and we are far from having an ontologically well-founded representation language we can reason with. Moreover, nobody has explored, as far as I am aware of, the computational impact of a representation language whose semantics is constrained in the light of ontological distinctions.

In the following, I will revisit the motivations and the main proposal of my old 1994 paper [17] in the light of the main results achieved so far, arguing for the need of further work[1]. This paper is organized as follows. In the next section I will discuss the very notion of "levels" for knowledge representation languages, based on a classic paper by Ron Brachman [4], and I will argue in favor of the introduction of a specific ontological level. Then, in Section 3, I will present examples showing the practical necessity of an explicit ontological commitment for representation constructs. In section 4, I will recap the recent results concerning formal ontological distinctions

[1] Most of the material presented here has been used in PhD courses on "Foundations of Conceptual Modeling and Ontological Analysis" John Mylopoulos and I have been giving for a couple of years (with slight changes in focus) at the ICT International School of the University of Trento. The idea was to present our own approaches in a complementary way, being both present throughout the course and making comments on each other's lectures on the fly. A lot of fun.

among unary and binary relations, sketching a basic ontology of meta-level categories representation languages should be aware of. In section 5, I discuss the role of the ontological level in current practice of knowledge engineering.

2 Knowledge Representation Levels

In 1979, Ron Brachman discussed a classification of the various primitives used by KR systems at that time [4]. He argued that they could be grouped in four levels, ranging from the implementational to the linguistic level (Fig. 2). Each level corresponds to an explicit set of primitives offered to the knowledge engineer. At the *implementational level*, primitives are merely pointers and memory cells, which allow us to construct data structures with no a priori semantics. At the *logical level*, primitives are propositions, predicates, logical functions and operators, which are given a formal semantics in terms of relations among objects in the real world. No particular assumption is made however as to the nature of such relations: classical predicate logic is a general, uniform, neutral formalism, and the user is free to adapt it to its own representation purposes. At the *conceptual level*, primitives have a definite cognitive interpretation, corresponding to language-independent concepts like elementary actions or thematic roles. Finally, primitives at the *linguistic level* are associated directly to nouns and verbs of a specific natural language.

Level	Primitives
Implementational	Memory cells, pointers
Logical	Propositions, predicates, functions, logical operators
Epistemological	Concept types, structuring relations
Conceptual	Conceptual relations, primitive objects and actions
Linguistic	Linguistic terms

Fig. 2. Classification of primitives used in KR formalisms (adapted from [4]). Epistemological level was “the missing level”.

Brachman’s KL-ONE [4,7] was the first example of a formalism built around these notions. Its main contribution was to give an epistemological foundation to cognitive structures like frames and semantic networks, whose formal contradictions had been revealed in the famous “What’s in a link?” paper [38]. Brachman’s answer to Woods’ question was that *conceptual links* should be accounted for by *epistemological links*, which represent the structural connections in our knowledge needed to justify conceptual inferences. KL-ONE focused in particular on the inferences related to the so-called IS-A relationship, offering primitives to describe the (minimal) formal structure of a concept needed to guarantee “formal inferences about the relationship (subsumption) between a concept and another”. Such formal structure consisted of the basic concept's constituents (primitive concepts and role expressions) and the constraints among them, independently of any commitment as to:

- the meaning of primitive concepts;
- the meaning of roles themselves;
- the nature of each role's contribution to the meaning of a specific concept.

The intended meaning of concepts remained therefore totally arbitrary: indeed, the semantics of current descendants of KL-ONE, description logics, is such that concepts correspond to *arbitrary* monadic predicates, while roles are *arbitrary* binary relations. In other words, at the epistemological level, emphasis is more on *formal reasoning* than on (formal) *representation:* the very task of representation, i.e. the structuring of a domain, is left to the user.

Current frame-based languages and object-oriented formalisms suffer from the same problem, which is common to all epistemological-level languages. On the one hand, their advantage over purely logical languages is that some predicates, such as those corresponding to types and attributes, acquire a peculiar, *structuring* meaning. Such meaning is the result of a number of ontological commitments, often motivated by strong cognitive and linguistic reasons and ultimately dependent on the particular task being considered, which accumulate in layers starting from the very beginning of the process of developing a knowledge base [11]. On the other hand, such ontological commitments remain hidden in the knowledge engineer's mind, since these knowledge representation languages are in general *neutral* as concerns ontological choices. This is also, in a sense, a result of the *essential ontological promiscuity* claimed by influential scholars [13, 27] for AI languages: since conceptualizations are our own inventions, then we need the maximum freedom for interpreting our representations.

Level	Primitive constructs	Main feature	Interpretation
Logical	Predicates	Formalisation	Arbitrary
Epistemological	Structuring relations (concepts and roles)	Structure	Arbitrary
Ontological	Structuring relations *satisfying meaning postulates*	*Meaning*	*Constrained*
Conceptual	Cognitive primitives	Conceptualisation	Subjective
Linguistic	Linguistic primitives	Language	Subjective

Fig. 3. Main features of the ontological level

In my 1994 paper I argued against this neutrality, claiming that a rigorous ontological foundation for knowledge representation can improve the quality of the knowledge engineering process, making it easier to build at least understandable (if not reusable) knowledge bases. After all, even if our representations are ontologically promiscuous, admitting the existence of whatever is relevant for us, it seems certainly useful to avoid at least the most serious ontological ambiguities when it comes to interpretation, by using different constructs for different basic ontological categories. In this view, as we shall see, "being large" and "being a rock" are represented by different constructs, whose semantics is constrained to reflect general ontological distinctions.

Representation languages conforming to this view belong to the *ontological level*, a new level I proposed to include in Brachman's layered classification, in an intermediate position between the epistemological and the conceptual levels (Fig. 3). While the epistemological level is the level of *structure*, the ontological level is the level of *meaning*. At the ontological level, knowledge primitives satisfy formal meaning postulates, which restrict the interpretation of a logical theory on the basis of formal ontological distinctions.

3 From the Logical Level to the Ontological Level

Suppose we want to state that a red apple exists. At the *logical level*, it is straightforward to write down something like

(1) $\exists x\ (\text{Apple}(x) \land \text{Red}(x))$.

At the epistemological level, if we want to impose some *structure* on our domain (dividing for instance apple from pears), the simplest formalism we may resort to is many-sorted logic. Yet, we have to decide which predicates correspond to sorts, as we may write

(2) $\exists x{:}\text{Apple}(\text{Red}(x))$

as well as

(3) $\exists x{:}\text{Red}(\text{Apple}(x))$

or maybe

(4) $\exists (x{:}\text{Apple}, y{:}\text{Red})(x=y)$.

All these structured formalizations are equivalent to the previous one-sorted axiom, but each contains an implicit structuring choice. However, (3) sounds intuitively odd: what are we quantifying over? Do we assume the existence of "instances of redness" that can have the property of being apples?

Unfortunately, the formalism we are using does not help us in making the right choice: we have the notion of "sort", but its semantics is completely neutral, since a sort may correspond to an *arbitrary* unary predicate. Using a more structured formalism allowing for attributes or (so-called) roles, like a description logic or a frame-based language, does not help, since we still have to make a choice between, say

(5) (a Apple with Color red)

and

(6) (a Red with Shape apple)

So, at the epistemological level, the structuring choices are up to the user, and there is no way to exclude the "unnatural" ones.

At the *ontological level,* on the contrary, what we want is a formal, restricted semantic account that reflects the ontological commitment underlying each structuring primitive, so that the association between a logical predicate and a structuring primitive is not a neutral choice any more: in other words, each structuring primitive corresponds to properties (or relations) *of a certain kind*. In our example, the difference between "being an apple" and "being red" lies in the fact that the former property supplies a principle for distinguishing and tracing in time its individual instances, while the latter does not. This distinction is known in the philosophical literature as the distinction between *sortal* and non-sortal (or *characterising*) properties [14], and is (roughly) reflected in natural language by the fact that the former are denoted by common nouns, while the latter by adjectives. The bottom line is that not all properties are the same, and only sortal properties correspond to what are usually called "concepts".

In the light of the above criteria, a predicate like *Red* – under its ordinary meaning – will not satisfy the conditions for being a concept (or a sort). Notice however that this may be simply a matter of point of view: at the ontological level, it is still the user who decides which conditions reflect the *intended* use of the *Red* predicate. For example, consider a different scenario for our example. Suppose there is a painter, who has a palette where the various colors are labeled with terms evoking natural things. For her, the various shades of red in the palette are labeled "orange red", "cherry red", "strawberry red", "apple red". In this scenario, the formula (3) above makes perfect sense, meaning that, among the various reds, there is also the apple red.

How can we account for such semantic differences? We shall see in the following that they are not simply related to the fact that the argument of *Red* belongs to different domains, but they reflect different *ways of predication,* expressed by predicates belonging to different *kinds*, in virtue of their different ontological nature. In part, these differences are also revealed by the way we use the same word in natural language: for instance, in the first scenario *Red* is an adjective, while in the painter's scenario it is a noun. Unfortunately this basic difference disappears when we move from linguistic analysis to logic analysis, since we tend to use the same predicate for the two cases.

In a knowledge representation formalism, we are constantly using natural language words within our formulas, relying on them to make our statements readable and to convey meanings we have not explicitly stated: however, since words are ambiguous in natural language, when these words become predicate symbols it may be important to "tag" them with an ontological category, endowed with a suitable axiomatization, in order to make sure the proper intended meaning is conveyed, and to exclude at least the most serious misunderstandings. This is basically what Chris Welty and I have suggested with our OntoClean methodology [21]. However, with my *ontological level* proposal, I was aiming at something more: embed some basic ontological categories in a knowledge representation formalism, constraining its own representation primitives. In part, this is what has been attempted by Giancarlo Guizzardi in his PhD work [24]. However, this work only concern semantic constraints on a conceptual modeling language (UML V2.0), and I am not aware of similar attempts for constraining the semantics of knowledge representation formalisms such as description logics.

In the following, I will briefly sum up and revisit the most relevant distinctions within unary properties and binary relations which have emerged from the research on formal ontology since the time I published my 1994 paper, and which I believe make sense from the point of view of knowledge representation. Hopefully, such distinctions will inspire a future generation of ontological level representation languages.

4 Basic Distinctions among Properties

In [19], Chris Welty and I presented a general ontology of unary properties, resulting from the combinatorial composition of a small set of *formal metaproperties* based on

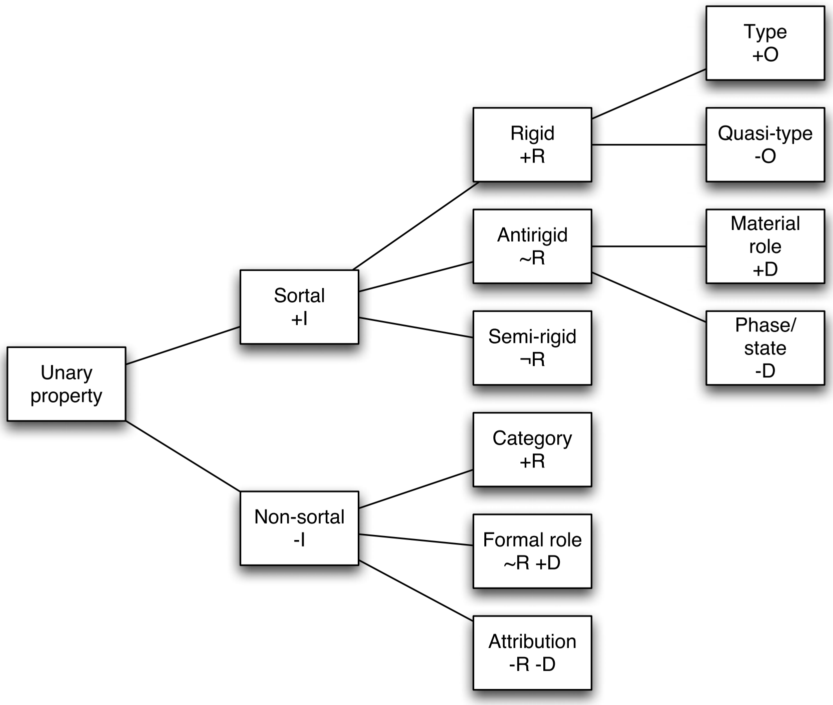


Fig. 4. A general ontology of unary properties. Adapted from [19].

three main notions: *identity, rigidity* and *dependence*, reported (in slightly revised form) in Fig. 4. I will not go here into the details of the technical aspects underlying these metaproperties, whose formal definitions have been discussed and refined in various papers since my early proposals [16, 17, 36, 31, 24]. I will just introduce them in an informal way as needed, pointing to the most recent formalizations.

What I would like to insist on here is the practical relevance of these distinctions: not all unary properties play the same role in knowledge representation, despite the fact that all of them can be expressed by the same logical structure (unary predicate).

Before introducing these property kinds, let me stress that they are *completely general*, being independent of any commitment concerning the ontological nature of the property arguments. In other words, the reason why a certain property belongs to one of these kinds has nothing to do with its arguments, which may belong – for instance – to any of the DOLCE's top categories like objects, events, or qualities.

4.1 Sortal vs. Non-sortal Properties

The first basic distinction is the classic one between *sortal* and *non-sortal* properties. In short, a property is a sortal (marked with the meta-property +I) if it carries a *criterion of identity* for its instances. Otherwise it is a non-sortal, marked with –I.

I will not enter here in the (still well alive, see [14]) philosophical debate related to the nature of sortals, simply claiming that, especially for knowledge representation purposes, it is *extremely useful* to distinguish between properties for which a certain principle for distinguishing and tracing their instances can be determined, and properties for which such principle cannot be determined[2]. Indeed, besides being well recognized in philosophy and in linguistics, the role of identity principles is explicitly defended in conceptual modeling (for instance, in Chen's Entity-Relationship model [10], entities are explicitly defined as "'things' which can be distinctly identified").

I only note here that, differently from [17] and [23] (but consistently with the OntoClean literature) I include non-countable properties corresponding to so-called *mass-terms* (like "amount of gold") under sortals. The rationale for this is that amounts of matter *can* indeed be distinguished and traced in time, differently from non-sortal properties like "red" (in the adjectival sense), and can appear in relative clauses instantiating the pattern "the X that …", such as "the amount of water that was in the glass is now on the floor". Indeed, assuming an atomic view of amounts of matter, their identity criterion is very simple: two amounts of matter are the same if and only if they contain the same molecules (similarly to collectives like groups of people). After all, we *need* to distinguish and trace amounts of matter if we want to model flow of liquids, for instance.

So being a sortal does not imply being countable, although the converse is true, at least for ordinary domains[3], and indeed countability is a useful heuristic to conclude that a property is a sortal, independently whether a particular identity criteria can be determined.

[2] See [20] for a formal account of the notion of identity criteria in knowledge representation.
[3] See [29] for an argument against the fact that countability implies identity.

4.2 Kinds of Rigidity

I introduced the first time the notion of ontological rigidity for a unary property in [16][4]. Since then, Chris Welty and I proposed a more careful definition in our Ontoclean papers [20, 21], which was further refined by various contributions [28, 8, 1, 36]. The basic intuition is however still the same: a unary property is rigid if it is essential for all its instances, so that, if x is an instance of a rigid property, it cannot lose this property without losing its identity. Going back to our example, it seems plausible to assume that *Apple* is always rigid (+R), while *Red* is non-rigid (-R) in the first scenario, and rigid in the painter's scenario. We see therefore how clarifying whether a property is rigid or not helps disambiguating between different ontological assumptions concerning the use of a certain word.

Since the definition of rigidity involves a universal quantification on all the instances of a given property, we can isolate two forms of rigidity: in the weaker case (non-rigidity, -R) there is at least one *contingent instance*, which does not exhibit the given property necessarily; in the stronger case (anti-rigidity, ~R), all instances are contingent. Of course anti-rigidity implies non-rigidity; a property which is non-rigid but not anti-rigid is called *semi-rigid* (¬R). As we shall see, *Student* is a classic example of an anti-rigid property (since every student is not necessarily such), while *Red* can be considered as semi-rigid, if we assume that certain things (say, rubies) are necessarily red, while others (e.g., red cars) are just contingently so. As shown in Fig. 4, sortals can be partitioned in rigid, anti-rigid and semi-rigid.

As stressed many times in the OntoClean papers, I would like to remark here that, in a certain KR theory, the decision as to whether a certain property is rigid or not is not a fixed one, and ultimately depends on the knowledge engineer: for example, if one believes in reincarnation, perhaps it makes sense to assume that *Person* is not rigid, if the worlds concerning the other lives are part of the modeling context. In a recent paper addressing again the definition of an ontology [22], I have elaborated this issue suggesting that a world is defined with respect to a specific observer (the knowledge engineer) and (forgetting time for the sake of simplicity) coincides with a maximal "perception state". So, for the knowledge engineering practice, rigidity only concerns those worlds that are in the modeler's radar.

4.3 Rigid Sortals: Types and Quasi-types

Rigid sortals are particularly important in knowledge engineering, since they capture the *essential, invariant aspects* of individuals, providing at the same time the criteria for individuating them in a given world, and tracing them across worlds. It seems very natural therefore, as introduced in [20] and further elaborated in [24], to impose, as modeling constraint, that *every element of the domain of discourse must be an instance of a rigid sortal*, complying to Quine's ditto "no entity without identity". Assuming this constraint, while analyzing a domain we can concentrate first on such rigid properties, forgetting the non-rigid ones, being assured that no domain elements are left out.

[4] I was unfortunately unaware of the work by Gupta [25], subsequently cited in [23] who introduced a very similar notion, called *modal constancy*.

Since rigid sortals can specialize each other, it is also useful to distinguish, within a sortals taxonomy, between those which just *carry* some identity criteria (inherited from some more general sortal) and those that directly *supply* the (necessary or sufficient) conditions that contribute to such criteria. We call the latter *types*, and the former *quasi-types*. According to the OntoClean notation, types are marked with the metaproperty +O, which stands for "supplies its own identity", and quasi-types with the metaproperty –O. For instance, consider the properties *Living Being*, *Person*, and *Italian Person.* Assuming that all of them are rigid, *Living Being* supplies some identity criteria (say, DNA identity), which are further specialized by *Person*, which adds*,* e.g, identity of fingerprints as a sufficient condition. Presumably, *Italian Person* does not supply further identity conditions, so the former two properties are types, while the latter is a quasi-type.

4.4 Anti-rigid Sortals: Material Roles and Phases

Since the early KL-One, the notion of role has been extensively discussed in the KR literature (see [3] for a recent overview). Various issues are still open, but there is a substantial agreement on the fact that unary properties denoting roles are anti-rigid. Anti-rigidity alone is however not enough to capture the relational nature of roles, which has been called *foundation* in [16], *external dependence* in [19], and again *foundation* in [31], always with slightly different formalizations. The latter formalization (which in turn relies on the notion of *definitional dependence*) is definitely the most accurate for our purposes, but I prefer to call it again *external dependence*, just because I find the term more intuitive. So, according to this revised definition, a property *P* is *externally dependent* (marked with +D) if its definition involves (at least) another property *Q* such that, for every instance *x* of *P*, there exists an instance *y* of *Q* which is *external* to *x*, in the sense that *x* is not a part of *y*, and *y* is not a part of *x*[5].

In conclusion, roles are anti-rigid, externally dependent unary properties[6]. Being anti-rigid, roles do not supply any identity criteria, which in most cases are inherited by the types they specialize (as in the prototypical example *Student*, which inherits the identity criteria of *Person*). However, there are certain general roles, like *Part*, or so-called *thematic roles* like *Patient* or *Theme,* which are not conceivably subsumed by any sortal, and hence they are not sortal themselves. Within roles, we distinguish therefore *material roles*, which (indirectly) carry some identity criteria (+I) from *formal roles*, which do not carry identity (-I).

Note that within material roles we also include properties like *Pedestrian* or *By-pass capacitor,* which linguistically behave differently from *Student* or *Son.* In [16] I called the latter *relational roles*, and the former *non-relational roles* (see next section).

As we have seen, roles are *externally dependent* properties, characterized by the +D metaproperty. If such metaproperty does not hold, and still we have an anti-rigid

[5] See [31] for the formal definition, which is based on a reification on the properties *P* and *Q*. See also [35] for a general discussion on this property reification move.

[6] See below for their systematic link to *binary* properties (so that *Student* is systematically linked with *Has-Student* or *Student-of*).

sortal, this is a case of a *phasal sortal,* whose prototypical example is *Baby*: if somebody is a baby, we cannot assume that anything else must necessarily exist, so *Baby* is not externally dependent, while clearly being an anti-rigid sortal. Note that phasal sortals also include *states* like *Tired* or *Happy*, assuming it is a sortal inheriting identity criteria from, e.g., *Animal.* The difference between phases and states should be however further analyzed[7].

4.5 Semi-rigid Sortals

Semi-rigid sortals have been called "mixins" in our OntoClean papers, but I prefer to avoid this term since it is used with different meanings in the object-oriented literature, as discussed in [23]. I don't think semi-rigid sortals have a special role in knowledge representation, although in some cases they may correspond to useful generalizations. They are reported here just for completeness.

4.6 Non-sortals: Categories, Formal Roles, and Attributions

The bottom part of Fig. 4 describes the remaining three cases in our taxonomy of unary properties, concerning the relevant distinctions within non-sortals. Note that our assumption that every individual must be an instance of a sortal implies that non-sortals correspond to abstract classes in the UML terminology, that is, they cannot have direct instances.

A first case is that of so-called *categories*, consisting of general properties like *Entity* or *Object*, which do not exhibit any *common* criterion of identity for their instances (for this reason they have been called *dispersive* in [26]). These are usually the topmost concepts in an ontology.

Formal roles have already been discussed, they are anti-rigid and externally dependent, but they carry no identity criteria. Note that also relational properties like *Interesting*, *Strange* or *On-the-table* fit under this class, although they don't look like roles, probably because they are not denoted by a name.

Finally, in OntoClean we called *attributions* all those non-sortal properties which are simply non-rigid and not externally dependent. This is a large class, which includes *Red* and *Big* as well as *Broken*. In DOLCE, I assume that these attributions reflect *qualitative states* of entities, resulting from the fact that a specific quality is classified in a certain region of a quality space [30].

4.7 The Rocks Example Revisited

Going back to our introductory examples, it is easy to conclude, in the light of the above discussion, that *Metamorphic rock, Igneous rock* and *Sedimentary rock* are the only *types* in the picture (we might want to call them *kinds*, terminological distinctions are a matter of taste, here). *Large rock* and *Grey rock* are semi-rigid sortals or perhaps phasal sortals (depending whether we admit that the same rock can change size or color), while *Pet metamorphic rock* is a material role.

---

[7] Perhaps phases – together with material roles – supply *local* identity criteria, differently from states.

5 Basic Distinctions among Binary Properties

Analogously to unary properties, useful distinctions can be drawn within binary properties, with the purpose of developing more "ontology aware" representation formalisms. Unfortunately, the results in this area, in comparison to what has been done for unary properties, are much more scattered, and I am not aware of any attempt to propose a general ontology like the one described above[8].

The main practical problem of binary relations, from the KR point of view, is still the one raised by Bill Woods in the example I mentioned in the introduction: how to distinguish between the relations which contribute to the *internal structure* of a concept and those which do not? Or, in other words, how to decide whether a piece of information should be modeled in terms of an attribute-value pair or in terms of a genuine relation?

I discussed this issue in [16], suggesting that attributes should be confined to *relational roles*, *qualities*, and *parts*. Intuitively, all these cases fit under the linguistic test suggested by Woods to check whether a binary relation A can be considered as an attribute for an individual X:

Y is a value of the attribute A of X if we can say that
Y is an A of X (or Y *is the A of X*)

Retrospectively, in the light of the most recent (yet scattered) work on the ontology of relations, I believe that the intuition behind the use of the *of* preposition to capture the notion of attribute lies in the ontological distinction between *internal* and *external* relations, which is intertwined with the distinction between *formal* and *material* relations[9]. The picture I have mind for binary relations is sketched in Fig. 5. I assume first a distinction between *formal* and *material relations* [15], where a formal relation yields just because of the very existence of its relata, while a material relation needs, so to speak, another "grounding" entity. Suppose, for example that John is older than Mary and John loves Mary; the *Older-than* relationship is a formal one, while the *Loves* relationship is a material one, since – besides the existence of John and Mary – it requires an extra entity, namely the *event* consisting of the love between John and Mary. I assume that all material relations are grounded on events, in DOLCE's sense[10].

Within formal relations, I distinguish between the internal and the external ones, depending whether there is an *existential dependence* relationship between the relata. The basic kinds of internal relationships I have in mind (all formalized in DOLCE) are parthood, constitution, quality inherence, and participation, shown in the figure. There are however some technical problems concerning parthood and constitution (which are shown with an asterisk), since, if we take time into account, a specific parthood or constitution relationship can be understood as an internal relation only if it holds *necessarily* (concerning therefore an essential part); otherwise, we cannot simply say that such relationship holds without specifying the time frame (i.e. the

[8] See [15] for a recent philosophical exploration of the ontology of binary relations.
[9] I know that for some authors these terminologies are equivalent.
[10] I know that this assumption may be too strong in some cases (e.g., for certain relations between events), but I believe it is robust enough for knowledge engineering purposes.

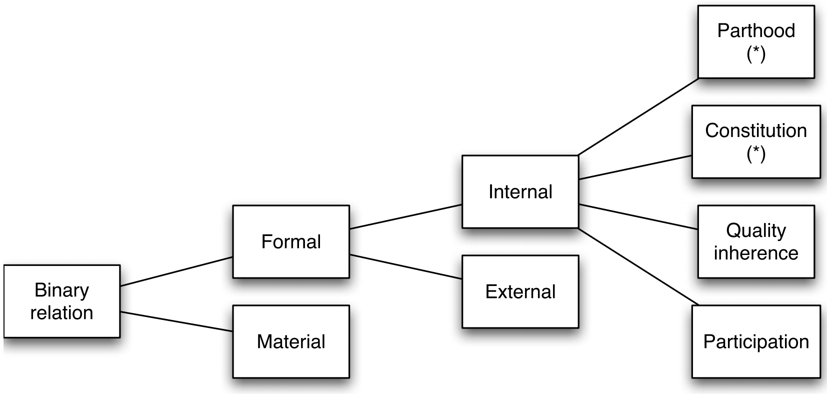


Fig. 5. A sketch of basic distinctions within binary relations

event) where this happens. I don't think that explicitly modeling events involving contingent parthood or constitution is a practical choice, however, so probably the best thing is to introduce suitable time-indexed parthood and constitution relations, whose formal characterization is still being investigated. However, my suggestion in the light of this analysis is that, in an ontologically well-founded theory, structuring relations (i.e., those corresponding to are called attributes or roles in frame-based formalisms and description logics) should be limited to specializations of such internal relationships, possibly extended with time indexes. This means that, for instance, an ownership relationship between a person and her car should be modeled in terms of the entity that grounds it, namely an event to which the person and the car participate. Similarly for the *Home Address* relation, which can be expressed in terms of the location of a *Dwelling* event.

In turn, such events can be modeled in terms of their own internal relations, including the various *participation relations* (thematic relations) expressing the various ways an object participates in an event. This systematic introduction of events in place of material relations may in some cases be excessively cumbersome, but in my opinion it is the only strategy that guarantees an explicit account of the modeler's ontological assumptions. Of course, if needed, more agile relations, such as ownership, can be *defined* in terms of this more basic picture.

6 Conclusions

I hope to have shown in this paper that in order to capture the *desiderata* for knowledge representation formalisms, as expressed in the old days and never properly met, it is necessary to formally express the ontological commitments of our representation constructs. This can be done in two ways:

1. by developing general ontologies built using ontologically neutral representation constructs,
2. by adopting *non-neutral* constructs, whose semantics is suitably constrained in order to guarantee ontologically well-founded models.

I believe that the second option is preferable, since it gives the knowledge engineer the tools to produce models with certain "guaranteed" properties in terms of ontological transparency, well-foundedness, and – therefore – reusability. In addition, I believe that reasoning with such constructs should be somewhat easier than with the first option, since the expressivity required to account for their ontological commitment belongs to the meta-language (i.e., the language used to account for the ontological semantics), and not to the object language. This is however an issue to be further investigated.

References

1. Andersen, W., Menzel, C.: Modal Rigidity in the OntoClean Methodology. In: Vieu, L., Varzi, A. (eds.) Formal Ontology and Information Systems: Collected Papers from the Fifth International Conference, pp. 119–127. IOS Press, Amsterdam (2004)
2. Artale, A., Franconi, E., Guarino, N., Pazzi, L.: Part-Whole Relations in Object-Centered Systems: an Overview. Data & Knowledge Engineering 20(3), 347–383 (1996)
3. Boella, G., van der Torre, L., Verhagen, H.: Roles, an Interdisciplinary Perspective. Applied Ontology 2(2), 81–88 (2007)
4. Brachman, R.J.: On the Epistemological Status of Semantic Networks. In: Findler, N.V. (ed.) Associative Networks: Representation and Use of Knowledge by Computers. Academic Press, London (1979)
5. Brachman, R., Fikes, R., Levesque, H.: Krypton: A Functional Approach to Knowledge Representation. IEEE Computer 116(10), 67–73 (1983)
6. Brachman, R., Levesque, H.: Competence in Knowledge Representation. In: Proceedings of National Conference on Artificial Intelligence (AAAI 1982), Pittsburgh, American Association for Artificial Intelligence, pp. 189–192 (1982)
7. Brachman, R.J., Schmolze, J.G.: An Overview of the KL-ONE Knowledge Representation System. Cognitive Science 9, 171–216 (1985)
8. Carrara, M., Giaretta, P., Morato, V., Soavi, M., Spolaore, G.: Identity and Modality in OntoClean. In: Vieu, L., Varzi, A. (eds.) Formal Ontology and Information Systems: Collected Papers from the Fifth International Conference, pp. 128–139. IOS Press, Amsterdam (2004)
9. Casati, R., Varzi, A.: Parts and Places. The Structure of Spatial Representation. MIT Press, Cambridge (1999)
10. Chen, P.: The entity-relationship model: Towards a unified view of data. ACM Transactions on Database Systems 1(1) (1976)
11. Davis, R., Shrobe, H., et al.: What is in a Knowledge Representation? AI Magazine (Spring 1993)
12. Fine, K.: Ontological Dependence. Proceedings of the Aristotelian Society 95, 269–290 (1995)
13. Genesereth, M.R., Nilsson, N.J.: Logical Foundation of Artificial Intelligence. Morgan Kaufmann, Los Altos (1987)
14. Grandy: Sortals. In: Zalta, E.N. (ed.) The Stanford Encyclopedia of Philosophy (2007)
15. Grenon, P.: On Relations. PhD Dissertation, Dept. of Philosophy, University of Geneve (2007)
16. Guarino, N.: Concepts, Attributes and Arbitrary Relations: Some Linguistic and Ontological Criteria for Structuring Knowledge Bases. Data and Knowledge Engineering 8, 249–261 (1992)

17. Guarino, N.: The Ontological Level. In: Casati, R., Smith, B., White, G. (eds.) Philosophy and the Cognitive Science, pp. 443–456. Hölder-Pichler-Tempsky, Vienna (1994)
18. Guarino, N., Carrara, M., Giaretta, P.: An Ontology of Meta-Level Categories. In: Sandewall, D.J.E., Torasso, P. (eds.) Principles of Knowledge Representation and Reasoning: Proceedings of the Fourth International Conference (KR 1994), pp. 270–280. Morgan Kaufmann, San Mateo (1994)
19. Guarino, N., Welty, C.: A Formal Ontology of Properties. In: Dieng, R., Corby, O. (eds.) EKAW 2000. LNCS, vol. 1937, pp. 97–112. Springer, Heidelberg (2000)
20. Guarino, N., Welty, C.: Identity and subsumption. In: Green, R., Bean, C., Myaeng, S. (eds.) The Semantics of Relationships: an Interdisciplinary Perspective, pp. 111–126. Kluwer, Dordrecht (2002a)
21. Guarino, N., Welty, C.: Evaluating Ontological Decisions with OntoClean. Communications of the ACM 45(2), 61–65 (2002b)
22. Guarino, N., Oberle, D., Staab, S.: What is an Ontology? In: Staab, S., Studer, R. (eds.) Handbook on Ontologies, 2nd edn., pp. 1–17. Springer, Heidelberg (2009)
23. Guizzardi, G., Wagner, G., Guarino, N., van Sinderen, M.: An Ontologically Well-Founded Profile for UML Conceptual Models. In: Persson, A., Stirna, J. (eds.) CAiSE 2004. LNCS, vol. 3084, pp. 112–126. Springer, Heidelberg (2004)
24. Guizzardi, G.: Ontological Foundations for Structural Conceptual Models. Telematica Instituut Fundamental Research Series 15 (2005)
25. Gupta, A.: The Logic of Common Nouns: An Investigation in Quantified Modal Logic. Yale University Press, New Haven (1980)
26. Hirsch, E.: Dividing Reality. Oxford University Press, New York (1993)
27. Hobbs, J.R.: Ontological Promiscuity. In: Proceedings of 23rd Annual Meeting of the Association for Computational Linguistics (ACL 1985), Chicago, IL, pp. 61–69 (1985)
28. Kaplan, A.: Towards a Consistent Logical Framework for Ontological Analysis. In: Welty, C., Smith, B. (eds.) Proceedings of FOIS 2001, pp. 244–255. IOS Press, Amsterdam (2001)
29. Lowe, E.J.: Entity, Identity, and Unity. Erkenntnis 48, 191–208 (1998)
30. Masolo, C., Borgo, S., Gangemi, A., Guarino, N., Oltramari, A., Schneider, L.: The WonderWeb Library of Foundational Ontologies and the DOLCE ontology. WonderWeb Deliverable, D17 (2002)
31. Masolo, C., Vieu, L., Bottazzi, E., Catenacci, C., Ferrario, R., Gangemi, A., Guarino, N.: Social Roles and their Descriptions. In: Proceedings of 9th Intl. Conference on the Principles of Knowledge Representation and Reasoning (KR 2004), Whistler, Canada, pp. 267–277 (2004)
32. Simons, P.: Parts: a Study in Ontology. Clarendon Press, Oxford (1987)
33. Varzi, A.: A note on the transitivity of parthood. Applied Ontology 1(2), 141–146 (2006)
34. Vieu, L.: On the transitivity of functional parthood. Applied Ontology 1(2), 147–155 (2006)
35. Vieu, L., Borgo, S., Masolo, C.: Artefacts and Roles: Modelling Strategies in a Multiplicative Ontology. In: Proceedings of FOIS 2008, Saarbruecken, Germany. IOS Press, Amsterdam (2008)
36. Welty, C., Andersen, W.: Towards OntoClean 2.0: A framework for rigidity. Applied Ontology 1(1), 107–116 (2005)
37. Winston, M., Chaffin, R., Herrmann, D.: A Taxonomy of Part-Whole Relations. Cognitive Science 11, 417–444 (1987)
38. Woods, W.A.: What's in a Link: Foundations for Semantic Networks. In: Bobrow, D.G., Collins, A.M. (eds.) Representation and Understanding: Studies in Cognitive Science. Academic Press, London (1975)

Some Notes on Models and Modelling

Michael Jackson

Department of Computing, The Open University, UK
jacksonma@acm.org

Abstract. Analytical models are a fundamental tool in the development of computer-based systems of every kind: their essential purpose is to support human understanding and reasoning in development. To support reasoning, models must be substantially formal. The relationship between a formal model and its—typically—non-formal subject demands care: particular attention must be paid to the model interpretation, which maps its formal terms to the phenomena of the subject. An analytical model is to be regarded not as an assertion, but as a predicate within a larger logical structure of reasoning. Analogical models, such as databases, act as run-time surrogates for some parts of the problem world; in their design the properties of the model itself must be carefully distinguished from those of its subject. Some models may be informal: informal models have many legitimate uses, but cannot serve as a basis for formal reasoning.

1 Modelling and Understanding

The subject of these notes is the use of models and modelling in the development of computer-based systems. Models are of many kinds, built for many purposes. Whatever the explicit purpose of a model, a vital implicit purpose is always to achieve, record and communicate some human understanding of its subject—that is, of whatever is being modelled. The subject and its understanding provide the central theme of these brief notes.

2 Models and Subjects

Richard Feynman, the physicist, described how as a teenager he had thought about certain elementary problems in Euclidean geometry [1]. He manipulated the diagrams in his mind: he anchored some points and let others float, imagined some lines as stiff rods and others as stretchable bands, and let the shapes slide until he could see what the result must be. He was using a mental model of a physical system to help him to understand an abstract mathematical system. Teachers of elementary arithmetic do something similar when they use a real physical model of an abstract mathematical system—for example, helping children to understand integer multiplication and division by playing with rectangular arrays of pennies. In both cases understanding of a less familiar *subject* is achieved through the medium of a more familiar, and therefore more accessible, *model*.

The word ‘model’ has many meanings and shades of meaning, but in the meanings that are most interesting in the development of software and information systems it always denotes one role in a binary relationship: the complementary role is 'subject'. The essence of this relationship is that in some way the model captures or expresses some understanding or knowledge of the subject, and can be used to provide further information or insight about it. As Feynman's pivoting rods and the arithmetic teacher's arrays of pennies show, there is no *a priori* requirement that the model be more abstract than the subject. It can be more concrete; but it must be in some way simpler, more accessible, more familiar, or more tractable than the subject. If it is not, the model achieves little or nothing: it can be discarded and the subject explored directly.

3 Mental Models and Reasoning

The model's superior accessibility, familiarity and tractability depend, of course, on the knowledge and experience of the people who construct and use it. An adult with a modest knowledge of arithmetic can dispense with the physical arrays of pennies, although they may still occasionally be useful in imagination to furnish a mental model. A number theorist will expect to think more abstractly, and perhaps more formally. Whether the number theorist is doing something radically different from forming and using a mental model appears to be an open question in cognitive science.

In *Mental models: a gentle guide for outsiders* [2] P N Johnson-Laird writes: "Our plan in the rest of this paper is to start with how models can be used to reason, and to contrast them with the orthodox view that reasoning is based on a sort of mental logic. ... On one side, there are those ... who claim that it depends on formal rules of inference akin to those of a logical calculus. On the other side, there are those, such as ourselves ..., who claim that it is a semantic process that depends on mental models akin to the models that logicians invoke in formulating the semantics of their calculi." It is, of course, perfectly plausible that human reasoning depends both on formal inference and on the use of mental models, combined according to the problem to be solved and the innate and learned capabilities, knowledge, and inclinations of the reasoner.

Johnson-Laird reports a reliable experimental difference between the following two questions. People find it harder to answer one of them correctly than the other:

"If A then B; A is true; what can be said about B?"

(to which the correct answer is "B is true") and

"If A then B; B is false; what can be said about A?"

(to which the correct answer is "A is false"). The second is the contrapositive of the first, and for someone with a little knowledge of elementary logic is of precisely equivalent difficulty. Yet the first question is answered correctly by nearly everyone, while a substantial minority of people fail on the second question, and those who do succeed take reliably longer to answer. Those who claim that reasoning depends on rules of inference identify a longer chain of deductions for the second question. Those who claim that it depends on mental models cite a *principle of truth*:

"Individuals tend to minimise the load on working memory by constructing mental models that represent what is true, but not what is false."

Evidently, human capacity for understanding is not itself easy to understand.

4 Overt Models

A *mental model* is a private possession, held in its owner's mind and revealed only at the owner's choice—and then only indirectly and uncertainly. An *overt model*, by contrast, is a public possession, intended to capture, communicate and make more widely available some understanding or notion of its subject, or some information about it.

Russell Ackoff distinguishes [3] three kinds of model used in natural science: iconic, analogue and symbolic. An *iconic* model looks like its subject. A photograph is an iconic model of its human subject, and a child's toy car is an iconic model of its subject automobile. An *analogue* model represents its subject by exhibiting different, but analogous, properties. The classic example is the use of a hydraulic system—water flowing in a complex of pipes—as an analogue model of an electrical circuit. The volume of water flowing, the narrowness of the pipes, and the water pressure are analogues of the electrical current, resistance, and potential difference respectively. Iconic models are always, to at least some small extent, also analogue models. A *symbolic* model, as its name implies, represents its subject symbolically, the symbols occurring as elements in some formalism that allows direct formal reasoning and calculation. Thus a system of equations may be a symbolic model. By solving the equations for given values of some variables the values of others can be determined. A symbolic model may be representable as an analogue model: for example, an equation may be represented by a graph.

In software development, which is a particular kind of engineering rather than a natural science, a slightly different taxonomy of models is useful. First, because engineering is concerned with specific artifacts and specific cases, we must distinguish *specific* from *generic* models. A commonly familiar example of the distinction is that between a class model and an instance model in object-oriented development. The class model is generic: its subject is those properties that are shared by all instances of the class. The instance model is specific: its subject is one instance only.

Second, the particular power of computers to create and maintain internal data structures make it both necessary and illuminating to distinguish *analytical* from *analogical* models. An analytical model is a *description* of its subject; in Ackoff's taxonomy it may be iconic or symbolic. For example, a finite state machine model of a possible behaviour of interest may be represented iconically in a diagram or symbolically in a transition table. Analytical models are most often used in the development process itself to help the developers to capture, understand and analyse properties of the problem world, the system requirements, and the software. A curve-fitting program, that chooses a curve type and adjusts the parameters of the curve to fit a set of data points, can be regarded as creating an analytical model at system execution time, but in most application areas this is exceptional. Because the properties of interest in software development are most often those that hold—or should hold—for all executions of the system, analytical models are typically generic. A finite state machine model, for example, may describe all successful withdrawals of cash from all possible ATMs in a particular system. Where some part of the problem world is a singleton and has only static properties—for example, the road layout for a particularly complex junction at which traffic is to be controlled—an appropriate analytical model may be specific rather than generic; but this is somewhat unusual.

An analogical model in software development is always specific. It is not used in the development process, but in the system operation: the analogical model is built and maintained by execution of the hardware/software machine. An analogical model is not a description of its subject, but a concrete thing in its own right. It is a data structure represented on a substratum of computer storage—most often as a collection of disk records or an assemblage of programming objects in RAM. The commonest example of such a specific analogical model is a database held on disk. The system continually maintains the database, using information directly input during system operation. The database can then act as an easily accessible *surrogate* for its subject, allowing the system to provide needed or requested information by examining the state of the database.

5 Model Imperfection

An overt analytical model is a description of its subject. The description may be formal or informal. It may be expressed in text, in equations, in diagrams, or in any other way judged suitable to the content of the description. An analytical model is inherently non-physical. It can be represented in a physical medium—for example, written on paper or encoded in a computer file; but the physical medium is not the model, just as the plastic disc of a CD is not the music. In a broad sense, this intangibility makes the model invulnerable to the vicissitudes of time and physical failure. The model can describe change over time, but it is not itself subject to change; it can describe physical decay and failure, but it is not itself subject to decay or failure.

However, this immunity to decay does not guarantee the quality of an analytical model *qua* model. For a non-formal subject, such as an engineering artifact, any analytical model—certainly at the levels of granularity of interest to software developers—is at best an approximation to its subject's actual physical behaviour and properties.

An analogical model, like an analytical model, is necessarily an imperfect approximation to its subject; but it is also imperfect in an additional way. Not only is the underlying understanding of the subject's properties inevitably an approximation, but the analogical model itself possesses phenomena and properties which have no counterpart in the subject and may distort understanding of the analogy. In the classic analogue model, we observe that a broken pipe leaks water: so we might—quite wrongly—infer that a broken wire will leak electricity into the air. In the same way, an analogical model incorporated into a computer-based system possesses phenomena and properties which have no counterpart in the subject. For example, a relational database may have null values in some row-column intersections; the rows may be ordered and indexed; and rows may be deleted for reasons of managing the database resources. These phenomena and properties can distort understanding of the analogy that underpins the relationship between model and subject.

6 Models and Interpretations

The meaning of the information provided by a model about its subject depends on an *interpretation*: that is, on an agreed mapping between the elements of the model and

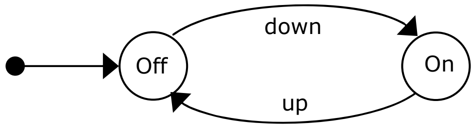


Fig. 1. An analytical model

the elements of the subject that they denote. Figure 1 shows an analytical model in the form of a state machine diagram having labelled circles for states and labelled arrows for transitions between states:

The arrow from the small solid circle points to the initial state. The interpretation must map the model's state labels to identifiable states of the physical subject, or its transition labels to identifiable events, or both. There are therefore at least these three components involved in using an analytical model: the subject matter; the model; and the interpretation.

The subject of this model may be the controlling switch of an electrical device. The interpretation may then be:

Off	≈	*state*: the switch is off
On	≈	*state*: the switch is on
down	≈	*event*: the switch is flipped down
up	≈	*event*: the switch is flipped up

The apparent simplicity of the model and interpretation may conceal some potential uncertainties. For example:

- The *On* and *Off* states may be independently observable phenomena of the subject. A more informative interpretation might then have been:

Off	≈	*state*: no current can flow through the switch
On	≈	*state*: current can flow through the switch

  Alternatively, it may be that the *On* and *Off* states are not independently observable phenomena of the subject, but are defined by the model:

Off	≝	*state*: either no *up* or *down* event has yet occurred, or else the most recent such event was an *up* event
On	≝	*state*: some *up* or *down* event has occurred, and the most recent such event was an *up* event

- The model may describe a switch, like an old-fashioned tumbler switch, in which two successive *up* events cannot occur without an intervening *down* event, and vice versa. Nothing is said about the effect of an *up* event in the *Off* state or a *down* event in the *On* state, because they cannot occur.

  Alternatively, the switch may be spring-loaded to return to a central position on each flip, placing no constraint on the sequence of *down* and *up* events. Nothing is said about the effect of an *up* event in the *Off* state or a *down* event in the *On* state, because they do not alter the switch state.

- Identifying a formal term with a physical event or state is in itself an abstraction. For example, treating *down* and *up* as atomic events abstracts from the complex physical processes they involve. In a tumbler switch, moving the knob compresses a spring as the knob moves towards the centre of its travel; when the knob passes the centre and moves to the end of its travel, the compressed spring exerts gradually increasing force on the switch contacts, eventually pulling them open and swinging them to their other position.

  Abstracting this process as an atomic event is a good choice if the switch operates fast enough and reliably enough for the purposes of the model being built.

- *Off*—whether defined or independently observable—is identified as the *initial* state, but the term *initial* has been given no interpretation. It may, for example mean:

```
Initial ≈ the state in which the newly manu-
          factured switch leaves the factory
```

  or:

```
Initial ≈ the state of the switch when the
          system begins execution
```

In a particular use of the model, the meaning of *Initial* may be given by the context in which the model is proposed. Whether their meanings are given by the model context, in an interpretation, or—as too often—left implicit, failure to deal properly with initial states is a rich source of error in software development. The problem of uninitialised variables is well-known in programming; but it is more difficult, and more often neglected, in the development of computer-based systems. The essence of the difficulty is to ensure compatibility between the initial state of the software to be executed and the current state of the problem world.

7 Designations

An element of an interpretation that associates a formal term in the model with a class of observable phenomena of the subject has been called [4] a *designation.* A designation must give a clear enough rule for recognising the phenomena to avoid harmful uncertainty. What is harmful depends on the nature and bounds of the subject, on the purpose to which the model will be put in the development, and on the opportunities that the developed system will offer for human common sense to override potential system failures resulting from modelling errors. In traditional manual systems, based on written processes and rules, system defects can often be repaired by reasonably resorting to available exception procedures when the system would otherwise deliver absurd results. To the extent that a computer-based system aims at automation it excludes such exception procedures. It is therefore important that developers' analytical models should correspond very closely to the subjects they describe.

A good correspondence between model and subject requires care in choosing and distinguishing the different subject phenomena to be designated. Consider, for example, a designation of the term *mother*:

```
mother(m,p) ≈ m and p are human beings and m
              is the mother of p
```

For a system concerned with Old Testament genealogy this designation is adequate: within the scope of the human beings mentioned in the Old Testament the meaning of "*m* is the mother of *p*" is perfectly clear. For a system to manage the administration of a kindergarten it may perhaps be good enough, provided that no special treatment is needed for adoptive mothers and stepmothers. In a fertility research clinic this designation would be useless: it would be necessary to distinguish natural mothers, genetic mothers, surrogate mothers and, perhaps, others.

If previously existing systems in the application area have a low degree of automation, the prevailing terminology of the area is not necessarily an adequate guide: application experts may underestimate the extent to which exceptional procedures are regularly invoked to compensate for terminological uncertainty. One well-known illustration is the phenomenon of a telephone *call*. Suppose that A phones B, but B is busy. A accepts the system's offer to connect them later, when B is no longer busy, and A hangs up. Soon, A's phone rings, and A picks it up and hears the ringback tone. After a short period of ringing, B answers and A and B talk. Is this one, two, or three *calls*? Reliance on the obsolete notion of a 'call' caused much difficulty in computer-based telephone systems in the last quarter of the twentieth century [5]. Similarly vague terminology is found in many application areas where the prevailing terminology includes obsolete relics of an earlier, much simpler, system.

8 Interpretation for Analogical Models

Interpretation for an analogical model is significantly more complex than for an analytical model. At first sight it may seem that the same notion of interpretation will serve for analogical as for analytical models: an interpretation maps the terms of the model to the phenomena of the subject. However, there is an important difference. An overt analytical model is itself a description, expressed in some chosen language, using a finite number of terms. An interpretation maps just those terms to the elements of the subject. Physical phenomena of any tangible representation of the model are to be ignored. For the state machine shown in Figure 1 we do not seek an interpretation of the lengths of the arrows or the diameter of the circles: the semantics of a description in the chosen graphical language are unaffected by those graphical phenomena.

An analogical model, by contrast, is a physical—and therefore inevitably non-formal—thing: it is a concrete structure of phenomena, not an abstract structure of formal terms. It does not embody any clear distinction between those of its own phenomena that are intended to participate in the analogy and those that are not. In effect, to understand the analogical model we also need an explicit analytical model.

Figure 2 shows how model, subject, and interpretation are related for a simple analytical model and for an analogical model.

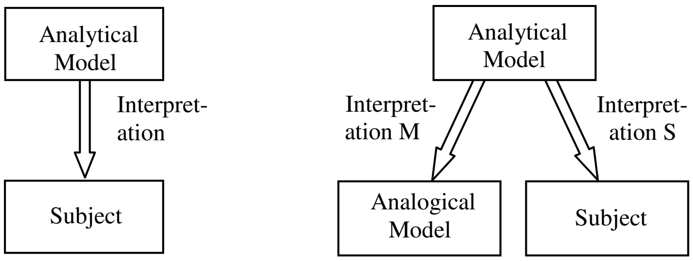


Fig. 2. Interpreting Analytical and Analogical Models

The purpose of the analytical model in the right side of Figure 2 is to bound the physical properties of interest in the analogical model, by pointing out the analogies that relate it to the subject. This analytical model therefore has two distinct interpretations: *S* interprets it as a model of the subject; the other, *M*, as a model of the analogical model. The analogy is in this sense indirect: an exact understanding rests on the analytical model that the analogical model shares with its own subject.

9 Designing and Understanding an Analogical Model

In developing any model—whether analytical or analogical—the conceptual starting point must be to consider what questions the model is intended to answer, and what properties it must therefore capture. In both cases, these are properties of the subject of the model, not of the model itself.

For an analytical model, the subject properties are directly expressed in the model. For an analogical model, the development process is conceptually more complex. First, the subject must be understood, and an appropriate analytical model developed. Then a type of analogical model—perhaps a database, or an assemblage of objects, or more generally a data structure in the programming language—is chosen or designed to offer exact analogues of the properties of the analytical model. Of course, there is a difficulty here. Because the analogical model is a concrete thing in its own right, and will inevitably possess properties that have no analogue at all in the subject, or impose constraints that are not present in the subject, the analogy is inevitably imperfect. So a part of the design task is to find good a compromise between fidelity to the subject and efficiency in the representations and accesses that the model affords. This task demands some clarity of thought: in particular, it demands a clear distinction between the properties of the model and the properties of the subject.

There is an obvious temptation to save time and effort by abbreviating the development task, short-circuiting the two interpretations and developing the analogical model directly from the subject. This is a common approach in object-oriented modelling, in which the developer describes the subject domain as if it were itself an assemblage of objects communicating by sending messages to each other. The benefit of the approach is a certain economy and directness. The disadvantage is an increased risk of error: the analytical model disappears from view, and is considered only tacitly and sporadically during the design of the analogical model.

10 The Context of a Model

A model has a purpose, in the sense that it is intended to capture certain properties that its subject may have, and to answer certain questions about them. But it also has a *context*, in a broader sense. For an analogical model in a computer-based system the context is, essentially, always the same. The computer continually collects information about the subject in the problem world, perhaps analyses and summarises it in some way, and updates the model to reflect it. The analogical model can then serve as a surrogate for the world: the state of the model is an acceptably accurate analogue of the state of the world. Its context is implicit: the presence of the model constitutes an assertion that the analogous properties hold in the subject.

For an analytical model, by contrast, there are many possible contexts. An analytical model can be thought of a predicate *M* applied to its subject *S*: *M(S)* holds if—and only if—the model is a true description of the subject. Just as a predicate can appear in a sentence, so too can an analytical model. For example, in a system to control a lift, the problem world *W* consists of the building's floors and lift shafts, the electro-mechanical lift equipment, the users, the request buttons and display lights, and so on. We can imagine three distinct analytical models of the problem world:

- *G(W)*: this model captures the *given* properties of the problem world. It describes, for example, the arrangement of the floors, the causal chain between the motor setting and the behaviour of the lift car, the way the lift and lobby doors work, and so on.
- *R(W)*: this model captures the *required* properties of the problem world, that the computer must somehow enforce. It describes, for example, the property that if the *Up* button at a floor is pressed, the lift car will eventually arrive at the floor and the doors will open, and that if a button inside the lift is then pressed that corresponds to a higher floor the lift car will go to that higher floor, and so on.
- *C(W)*: this model captures the behaviour at the *computer's* interface with the problem world. It describes, for example, the property that if the *Up* button at floor 3 is pressed when the lift is stationary at the ground floor, then the motor direction is set to *Up* and the motor is switched *On* (by the computer), that when subsequently the sensor at floor 3 is set *On* (by the arrival of the lift car) the motor is then switched *Off* (by the computer), and so on.

In a successful development, the relationship among these three models is something like:

$$(\mathrm{G(W)} \wedge \mathrm{C(W)}) \Rightarrow \mathrm{R(W)}$$

That is: the given properties of the problem world, in conjunction with the additional properties due to its interaction with the computer, ensure that the requirement is satisfied. The *truth* of each model depends on the changing context. At the outset of the development, *G(W)* is true (assuming that the building and the lift equipment are known and correctly described). At that point in time, however, *R(W)* nor *C(W)* is true. *C(W)* is not true because the computer has not yet been built and installed; and *R(W)* is also not true—in the absence of the computer it is merely what the customer wishes were true. Later, when the development has been successfully completed, and the software is executing as intended, all three models will be true.

11 The Local Context of a Model

The formula *(G(W* $\wedge$ *C(W))* $\Rightarrow$ *R(W)*, interpreted for a whole system, expresses a global context for the three models: it encompasses the whole development, and the behaviour of the whole system. In the course of the development it will be necessary, for a realistic system, to decompose all three models in a way that is not necessarily simple. These decompositions aim to master the complexity of the development problem and the eventually developed system, reducing one large and unmanageably complex problem to a structure of simpler problems.

The decompositions that will be most useful will depend on the problem in hand. Because realistic systems are complex in a heterogeneous way, the most useful decompositions will be similarly heterogeneous. Some candidate dimensions of decomposition are:

- Decomposition by function or feature. An e-commerce system for consumer use has such features as shopping basket, credit card validation and charging, collaborative filtering, shipping management, and so on.
- Decomposition by operational phase. An avionics system must behave differently in the different phases of a flight: in taxiing, taking off, climbing, cruising and so on.
- Decomposition by problem world conditions. The behaviour of an air traffic control system in normal conditions is different from its behaviour in an emergency. A lift control system providing normal lift service must behave differently when an equipment fault—such as a failing hoist motor—is detected.

Decomposition can be usefully regarded as decomposition into subproblems, in which each subproblem defines a local context. Different subproblems have different requirements; they need different software behaviours for their solution; they concern different parts or *domains* of the problem world; and they exploit different properties of those domains. To understand and analyse a subproblem it is necessary to practise a separation of concerns. The models needed for a subproblem are local models for local contexts.

The local context of a model puts in place a set of local assumptions, determining its purpose and the content of the description it embodies of its subject. In a lift system, for example, the local context of providing normal lift service assumes that the equipment functions normally, exhibiting the behaviour that is necessary if the lift car is to be sent from floor to floor, the doors opened and closed appropriately, and so on. The relevant model of the lift equipment is therefore a model of *healthy* lift equipment, describing the normal functioning. By contrast, the local context of fault monitoring requires a model of *dubious* lift equipment, describing equipment that may or may not function normally, and focusing on the properties that allow faults to be detected and perhaps diagnosed, when they occur.

12 The Scope and Span of a Model

The *scope* of a model is the set of all phenomena denoted by its interpreted terms. The scope of the model of the control switch shown in Figure 1 is {*up, down, On, Off*} if they are all designated, independently observable, phenomena; but if *On* and *Off* are defined in terms of *up* and *down*, then the scope is only {*up, down*}. The switch may have other phenomena—for example, it may have a rotary dimmer knob—but they are *out of scope*: the developer of the model has decided that for the purpose of the model they should be ignored.

The *span* of a model is how much it describes, measured by time or space or any other relevant quantifiable dimension. For example, in a lift control system it may be appropriate to model the required opening and closing of the lift doors in normal operation. One required property may be the opening and closing of the doors when the lift car serves a floor, including the behaviour when an obstruction is detected: a model of this property has a span of one visit to one floor by one lift. Another required property may be that the lift doors are never opened when the car is not positioned at a floor: a model of this property has a span of one lift over the whole local context of normal operation. A model whose span is one lift can, of course, be applied to all the lifts in the system; and a model whose span is one visit to one floor can be applied to all visits to all floors; the spans of the models themselves, however, are not affected by their larger application.

To be intelligible, the scope and span of a model must be appropriate to its subject and content. A notable—and widely practised—way of obfuscating a model is to replace it by a loose collection of models of inappropriately small span. For example, a state machine may be fragmented into a distributed collection of individual transitions. Each transition makes understandable sense only as a part of the whole state machine: taken alone it can be understood only by a reader who already possesses a firm and clear mental model of the whole machine, and has this mental model vividly in mind while reading each fragment. The obvious danger is that neither the writer nor the readers have such a mental model available, and the fragments are never brought together to validate their collective meaning. The result is a model prone to many errors. For example: omitting transitions that should be included; making false assumptions about reachability; and ignoring the disastrous effect of a neglected, but possible, sequence of transitions.

The appropriate span for a model is not always obvious. Whenever the behaviour of a system feature or function is arranged in sporadic or cyclic episodes—for example, in use cases—it is naturally attractive to construct a model of the function with a span of one episode. For some aspects of the episode this will be entirely appropriate. Much of the interaction of a bank customer with an ATM is encapsulated within the episode, and should be modelled with that span: the card is inserted before the PIN is entered; the card is withdrawn before the money is delivered; and so on. However, the episode may include events of a more global import: a certain amount of money is withdrawn from the account; a new PIN is specified; a new cheque book is requested. These events belong to behaviours of spans larger than the episode. Depending on the complexity of these behaviours, it may be necessary to model them also, each in its appropriate span.

Using an appropriate span for a model is not just a matter of bringing together enough information in one document. A good model answers its designed questions in the simplest possible way, laying the smallest possible burden on the reader's powers of perception, memory and reasoning, and helping the reader to form a good mental model of the subject. Because short-term human memory is severely limited, this process of assembling a mental model must not require information to be collated from many separate places. From a purely formal point of view, a graph may be equally well represented in a diagram or in a list of nodes and arcs; but from the point of view of human intelligibility the diagram is hugely superior for almost every purpose. The most important questions to be answered by a graph model are not usually questions about isolated nodes or arcs: they are questions about traversals of paths in the graph. The primary significance of a node or arc is its role and position in a set of possible traversals. The core success of the famous London Underground map is that it makes traversals—train journeys—very easy to identify. The importance of traversals explains also why even those computer scientists who disdain diagrams prefer to write their programs as structured, indented texts.

13 Vague Models

Almost any overt description—diagrammatic, textual or numerical—can be regarded as a generic or specific analytical model of its subject. Whether it is a useful model for its intended purpose will depend on many considerations. The importance of explicit designation for ensuring that the meanings of the terms used in the model are clear and exact has been stressed in this essay; but a shopping list can be useful even if some its entries are vague: "Lots of oranges if they're sweet", or even "Something nice for our dinner". A more structured checklist can be useful even if it is similarly vague: "The chief quality measures are high customer satisfaction and a low rate of operator error". The shopper and the project manager know that the terms in their models are very imprecise, and are careful not to lay too much weight on them. It is sometimes—but not always—worthwhile to establish quantitative empirical criteria for these imprecise terms.

A model can be vague for other reasons than the absence of designations of its terms. The descriptive language itself may use fuzzy general notions like "about" and "some" and "low rate"; some linguistic terms—such as "nice" and "satisfaction" may have no clear meaning that can be designated; some operators or connectives in the language—such as the lines or arrows in a graph, or the various node symbols—may have no clear semantics. This kind of vagueness can be easily tolerated, and can even be helpful, in contexts in which the model plays the role of a personal reminder, a sketch for live discussion, or an informal private note between two people; but it is very damaging if the model's purpose is to serve as a basis for any kind of reasoning. Any conclusions reached by reasoning about a model must be encashed by interpreting them in terms of the phenomena and relationships of the subject: if this cannot be done reliably then the value of the conclusions is diminished accordingly.

14 Building Precision on Vagueness

Formal reasoning cannot be based on an informal model. A faulty map cannot be corrected until two sources of faults have been eliminated. First, the cartographic conventions must be clearly established—for example, whether a road bridge over a railway is distinguished from a rail bridge over a road, and, if so, how. Second, the designations must be clarified—for example, does a cross signify only a church or can it also signify a mosque or synagogue? Then the map can be corrected by comparing it with the terrain it is intended to describe and modifying the map to correspond to the terrain. Similarly, the informality of a model in software development cannot be repaired without repairing the inadequacies both of the modelling language and of the designations that relate the model to its subject.

Suppose, for example, that a model is concerned with relationships of *dependency* among a population of distinct specific things or tasks or goals or documents. Such a model may be useful in program design, in tracing the relationship of an implementation to its requirements, and in other contexts too. In program design a relevant designation might be:

depends(m,n)	≈	*m* and *n* are program modules and *m* depends on *n* in the sense that *m* will not function correctly unless *n* functions correctly

This designation may seem clear enough. The writer or reader of the model may even be tempted to infer that depends is transitive:

$$\forall\ m,n,o \bullet \mathrm{depends}(m,n) \land \mathrm{depends}(n,o) \Rightarrow \mathrm{depends}(m,o)$$

However, perhaps the designation is far from clear enough. Suppose that modules in this context are procedures, that interaction is procedure call, and that a module *m* has functioned correctly if the result of '*m(p1,p2,...)*', including any side effects, satisfies the specification of *m*. Then to clarify the meaning of *depends(m,n)* it may be necessary to consider these and other possibilities:

- When *m* is called, it may, or may not, call *n* before returning to its caller.
- When *m* is called, it always calls *n* before returning to its caller.
- When *m* is called, it always calls *n* before returning to its caller, but *m* does not use any result of the call (for example, *n* simply logs the call).
- For some calls of *n*, *n* fails to satisfy its specification, but none of the calls for which it fails can be a call by *m*.
- *n* calls *m*, and *m* can satisfy its specification only if *n* executes an appropriate sequence of calls with appropriate arguments (for example, *m* is an output module encapsulating a file and requiring the sequence of calls

```
<m('open'); m('write',v)*; m('close')> ).
```

- *m* and *n* both call a third module *q*, and *q* can satisfy its specification only if it is called by an appropriate sequence of calls with appropriate arguments, calls by *m* and calls by *n* being interleaved.

It may be possible—or impossible—to provide a clear designation of *depends(m,n)* in the particular subject to be modelled. If it is impossible, there is no point is building an edifice of formal reasoning on such shaky foundations. It is not required that every useful model be formal and exact; but the writer and reader of a vague or informal model should avoid the mistake of treating it as a basis for formal reasoning.

15 Summary

These notes have briefly discussed several aspects of models and modelling in the context of software development. Their unifying theme is a pair of relationships in which any model must participate. One relates the overt model to the human understanding—that is, to the mental model—that it seeks to express or evoke. The other relates the overt model to its subject matter in the physical and human world. In effect, these two relationships unite to form a bridge between the world and our understanding of it. An effective practice of modelling must seek to create a bridge that is strong at both ends: it must find abstractions of reality that are adequate for the purpose in hand and its context; and it must express and convey those abstractions in ways that serve the goal of human understanding as well as possible.

Because computer programs are, in effect, formal and exact processes, they admit no vagueness in their execution. We must therefore be doubly confident in the formal models of the world on which we base our software development. As John von Neumann pointed out [6]:

"There is no point in using exact methods where there is no clarity in the concepts and issues to which they are to be applied."

References

1. Gleick, J.: Genius: Richard Feynman and Modern Physics. Little Brown (1999)
2. Johnson-Laird, P.N., Girotto, V., Legrenzi, P.: Mental models: a gentle guide for outsiders. Sistemi Intelligenti (1998)
3. Ackoff, R.L.: Scientific Method: Optimizing Applied Research Decisions. Wiley, Chichester (1962)
4. Jackson, M.: Software Requirements & Specifications: A Lexicon of Practice, Principles, and Prejudices. Addison-Wesley, Reading (1995)
5. Zave, P.: Calls Considered Harmful and Other Observations: A Tutorial on Telephony. In: Margaria, T., Steffen, B., Rückert, R., Posegga, J. (eds.) ACoS 1998, VISUAL 1998, and AIN 1997. LNCS, vol. 1385, pp. 8–27. Springer, Heidelberg (1998)
6. von Neumann, J., Morgenstern, O.: Theory of Games and Economic Behaviour. Princeton University Press, Princeton (1944)

A Semantical Account of Progression in the Presence of Defaults

Gerhard Lakemeyer[1] and Hector J. Levesque[2]

[1] Dept. of Computer Science
RWTH Aachen
52056 Germany
gerhard@cs.rwth-aachen.de
[2] Dept. of Computer Science
University of Toronto
Toronto, Ontario
Canada M5S 3A6
hector@cs.toronto.edu

Abstract. In previous work, we proposed a modal fragment of the situation calculus called $\mathcal{ES}$, which fully captures Reiter's basic action theories. $\mathcal{ES}$ also has epistemic features, including only-knowing, which refers to all that an agent knows in the sense of having a knowledge base. While our model of only-knowing has appealing properties in the static case, it appears to be problematic when actions come into play. First of all, its utility seems to be restricted to an agent's initial knowledge base. Second, while it has been shown that only-knowing correctly captures default inferences, this was only in the static case, and undesirable properties appear to arise in the presence of actions. In this paper, we remedy both of these shortcomings and propose a new dynamic semantics of only-knowing, which is closely related to Lin and Reiter's notion of progression when actions are performed and where defaults behave properly.

Preamble

A long time ago, John Mylopoulos' main area of research was knowledge representation. His work in this area included the PSN representation system, part of a larger effort intended to provide a natural-language front-end to databases (Mylopoulos, 1975). In the late seventies, John suggested that we investigate a small extension to PSN to allow for knowledge bases with *incomplete knowledge*. He realized that merely extending a classical true/false semantics to include a third value for *unknown* did not do the job. Among other things, tautologies could come out unknown.[1] What sort of semantic account would assign *unknown* to just those formulas whose truth values really were unknown?

Our attempt to answer this question in a satisfactory way has led us to an enormous amount of research, including both of our doctoral theses at the University of Toronto

[1] If a sentence p is unknown, then its negation $\neg p$ is also unknown. Then if, as is typical of three-valued logics, the disjunction of two unknown sentences is itself unknown, the sentence $(p \vee \neg p)$ would be unknown.

and a technical monograph. We investigated languages where you could distinguish between saying that a formula was *true* and that a formula was *known*. In the end, we came up with the idea of *only knowing*: saying that a formula (and everything that followed from it, either logically or through introspection) was all that was known. In the present paper, we continue this line of research and show how only-knowing should work in a *dynamic* setting, in particular, within a fragment of the situation calculus, as presented by Ray Reiter for reasoning about action and change. A short version of this chapter appeared in (Lakemeyer and Levesque, 2009).

1 Introduction

In previous work, we proposed a modal fragment of the situation calculus called $\mathcal{ES}$, which fully captures Reiter's basic action theories and regression-based reasoning, including reasoning about knowledge (Lakemeyer and Levesque, 2004; Lakemeyer and Levesque, 2005). So, for example, the language allows us to formulate Reiter-style successor state axioms such as this one:

$$\begin{aligned}&\forall a, x.\Box([a]\mathit{Broken}(x) \;\equiv\\ &\quad (a = \mathit{drop}(x) \wedge \mathit{Fragile}(x)) \;\vee\\ &\quad (\mathit{Broken}(x) \wedge a \neq \mathit{repair}(x)))\end{aligned}$$

In English: after any sequence of actions ($\Box$), an object x will be broken after doing action a ($[a]\mathit{Broken}(x)$) iff a is the dropping of x when x is fragile, or x was already broken and a is not the action of repairing it. Here we assume that *Fragile* is a predicate which is not affected by any action so that its successor state axiom would be

$$\forall a, x.\Box([a]\mathit{Fragile}(x) \;\equiv\; \mathit{Fragile}(x)).$$

Let us call the conjunction of these two axioms SSA_{BF}. In addition to action and change, the language $\mathcal{ES}$ also addresses what an agent knows and only-knows. The latter is intended to capture all an agent knows in the sense of having a knowledge base. For illustration, consider the following sentence, which is logically valid in $\mathcal{ES}$:

$$\begin{aligned}&\boldsymbol{O}(\mathit{Fragile}(o) \wedge \neg\mathit{Broken}(o) \wedge \mathit{SSA}_{BF}) \;\supset\\ &\qquad [\mathit{drop}(o)]\,(\boldsymbol{K}(\mathit{Broken}(o)) \wedge \neg\boldsymbol{K}(\mathit{Glass}(o)))\,.\end{aligned}$$

In English: if all the agent knows is that *o* is fragile and not broken and that the successor state axioms for *Broken* and *Fragile* hold, then after dropping *o*, the agent knows that *o* is broken, but does not know that *o* is made of glass.

Let us now consider what the agent should only-know after the drop action has occurred. Intuitively, the agent's knowledge should change in that it now believes that *o* is broken, with everything else remaining the same. Formally,

$$[\mathit{drop}(o)]\,\boldsymbol{O}(\mathit{Fragile}(o) \wedge \mathit{Broken}(o) \wedge \mathit{SSA}_{BF}).$$

In fact this view corresponds essentially to what Lin and Reiter (LR) [1997] call the *progression* of a database wrt an action. It turns out, however, that the semantics of only-knowing as proposed in (Lakemeyer and Levesque, 2004) differs from this in that the last formula above is *not* entailed. The reason is that their version, unlike progression, does not forget what was true initially (like whether or not *o* was already broken), and so more ends up being known.

The LR notion of progression allows for efficient implementations under certain restrictions (Lin and Reiter, 1997; Liu and Levesque, 2005; Vassos and Levesque, 2007), and being able to forget the past seems essential for this. Hence the previous semantics of only-knowing may not be very useful, except perhaps in the initial state. In this paper, we present a new semantics of only-knowing which avoids this pitfall and is fully compatible with LR's idea of progression.

It was shown in (Levesque, 1990) that only-knowing in the static case also accounts for default reasoning in the sense of autoepistemic logic (Moore, 1985). For example, the default that objects are fragile unless known otherwise can be written as

$$\forall x \neg \boldsymbol{K} \neg \mathit{Fragile}(x) \supset \mathit{Fragile}(x).$$

If the agent uses this default instead of the fact that *o* is fragile then it would still conclude, this time by default, that *o* is fragile and hence believe that it is broken after dropping it. But suppose that *o* is actually *not* fragile. What should the agent believe after *sensing* the status of *o*'s fragility? Clearly, it should then believe that *o* is indeed not fragile and it should not believe that dropping *o* will break it. That is, the default should no longer apply. Unfortunately, the previous definition of only-knowing does not do this. The problem, roughly, is that the initial default conclusion that *o* is fragile cannot be distinguished from a hard fact. Subsequently sensing the opposite then leads to an inconsistency.

In this paper we will fix this problem by proposing a semantics which separates conclusions based on facts from those based on defaults. To this end, we will distinguish between what is known for sure (using the modality $\boldsymbol{K}$) and what is believed after applying defaults (using another modality $\boldsymbol{B}$). In fact, defaults themselves will be formulated using $\boldsymbol{B}$ instead of $\boldsymbol{K}$. All this will be integrated with progression in the sense that defaults will be applied to the progressed knowledge base.

The rest of the paper is organized as follows. In the next section, we introduce the logic $\mathcal{ES}_{\text{O}}$, which is like the old $\mathcal{ES}$ except for the new semantics of only-knowing and defaults. This semantics agrees with the previous one in the static case. After that, we consider only-knowing in the context of basic action theories. In particular, we show that what is only-known after an action extends LR's original idea of progression, and how reasoning about defaults fits into the picture. We then address related work and conclude.

2 The Logic $\mathcal{ES}_{\text{O}}$

The language is a second-order modal dialect with equality and sorts of type object and action. Before presenting the formal details, here are the main features:

- *rigid terms*: The ground terms of the language are taken to be isomorphic to the domain of discourse. This allows first-order quantification to be understood substitutionally. Equality can also be given a very simple treatment: two ground terms are equal only if they are identical.

– *knowledge and truth*: The language includes modal operators $\boldsymbol{K}$ and $\boldsymbol{B}$ for knowledge and belief. The $\boldsymbol{K}$ operator allows us to distinguish between sentences that are true and sentences that are known (by some implicit agent). The $\boldsymbol{B}$ operator allows an agent to have false beliefs about its world or how its world changes. For example, we can model situations where an object is not fragile but the agent does not know it, yet may believe that it is fragile by default.
– *sensing*: The connection between knowledge and truth is made with sensing. Every action is assumed to have a binary sensing result, like whether an object is fragile or not, and after performing the action, the agent learns that the action was possible (as indicated by the *Poss* predicate) and whether the sensing result for the action was 1 or 0 (as indicated by the *SF* predicate).[2] Just as an action theory may contain precondition axioms characterizing the conditions under which *Poss* holds, it can contain axioms characterizing the conditions under which *SF* holds.

2.1 The Language

Definition 1. *The symbols of* $\mathcal{ES}_0$ *are taken from the following vocabulary:*

– *first-order variables:* $x_1, x_2, \ldots, y_1, y_2, \ldots, a_1, a_2, \ldots$*;*
– *second-order predicate variables of arity k:* $P_1^k, P_2^k, \ldots$*;*
– *rigid functions of arity k:* $G^k = \{g_1^k, g_2^k, \ldots\}$*; for example, o, drop;*
– *fluent predicate symbols of arity k:* $F_1^k, F_2^k, \ldots$*; for example, Broken; we assume that this list includes the special predicates Poss and SF (for sensing);*
– *connectives and other symbols:* $=$, $\wedge$, $\neg$, $\forall$, $\boldsymbol{K}$, $\boldsymbol{B}$, $\boldsymbol{O}$, $\boldsymbol{\Omega}$, $\square$, *round and square parentheses, period, comma.* $\boldsymbol{K}$, $\boldsymbol{B}$, $\boldsymbol{O}$, *and* $\boldsymbol{\Omega}$ *are called epistemic operators.*

Definition 2. *The* terms *of the language are the least set of expressions such that*

1. *Every first-order variable is a term;*
2. *If* $t_1, \ldots, t_k$ *are terms, then so is* $g^k(t_1, \ldots, t_k)$.

We let $\mathcal{R}$ denote the set of all rigid terms (here, all ground terms). For simplicity, instead of having variables of the *action* sort distinct from those of the *object* sort as in the situation calculus, we lump both of these together and allow ourselves to use any term as an action or as an object.[3]

Definition 3. *The* well-formed formulas *of the language form the least set such that*

1. *If* $t_1, \ldots, t_k$ *are terms, and* F *is a* k*-ary predicate symbol then* $F(t_1, \ldots, t_k)$ *is an (atomic) formula;*
2. *If* $t_1, \ldots, t_k$ *are terms, and* V *is a* k*-ary second-order variable, then* $V(t_1, \ldots, t_k)$ *is an (atomic) formula;*

[2] For convenience, we assume that every action returns a (perhaps trivial) sensing result. Here, we restrict ourselves to binary values. See (Scherl and Levesque, 2003) for how to handle arbitrary sensing results.

[3] Equivalently, the version in this paper can be thought of as having action terms but no object terms.

3. *If t_1 and t_2 are terms, then $(t_1 = t_2)$ is a formula;*
4. *If t is a term and α is a formula, then $[t]\alpha$ is a formula;*
5. *If α and β are formulas, v is a first-order variable, and V is a second-order variable, then the following are also formulas: $(\alpha \wedge \beta)$, $\neg\alpha$, $\forall v.\, \alpha$, $\forall V.\, \alpha$, $\Box\alpha$, $\boldsymbol{K}\alpha$, $\boldsymbol{B}\alpha$, $\boldsymbol{O}\alpha$, and $\boldsymbol{\Omega}\alpha$, where the formulas following $\boldsymbol{O}$ and $\boldsymbol{\Omega}$ are restricted further below.*

We read $[t]\alpha$ as "α holds after action t", and $\Box\alpha$ as "α holds after any sequence of actions," and $\boldsymbol{K}\alpha$ ($\boldsymbol{B}\alpha$) as "the agent knows (believes) α." $\boldsymbol{O}\alpha$ may be read as "the agent only-knows α" and is intended to capture all the agent knows about what the world is like now and how it evolves as a result of actions. Here no defaults are taken into account, just facts which, as we will see later, come in the form of a basic action theory similar to those proposed by Reiter (2001). Therefore, we restrict $\boldsymbol{O}$ to apply to so-called *objective formulas* only, which are those mentioning no epistemic operators. Finally, $\boldsymbol{\Omega}\alpha$ is meant to capture all and only the defaults believed by the agent. For that, α is restricted to what we call *static belief formulas*, which mention neither $\Box$ nor $[t]$ nor any epistemic operator except $\boldsymbol{B}$.

As usual, we treat $(\alpha \vee \beta)$, $(\alpha \supset \beta)$, $(\alpha \equiv \beta)$, $\exists v.\, \alpha$, and $\exists V.\, \alpha$ as abbreviations. We use α^x_t to mean formula α with all free occurrences of variable x replaced by term t. We call a formula without free variables a *sentence*.

We will also sometimes refer to *static objective formulas,* which are the objective formulas among the static belief formulas, and *fluent formulas*, which are formulas with no $\boldsymbol{K}$, $\boldsymbol{O}$, $\boldsymbol{B}$, $\boldsymbol{\Omega}$, $\Box$, $[t]$, *Poss*, or *SF*.[4]

2.2 The Semantics

The main purpose of the semantics we are about to present is to be precise about how we handle fluents, which may vary as the result of actions and whose values may be unknown. Intuitively, to determine whether or not a sentence α is true after a sequence of actions z has been performed, we need to specify two things: a world w and an epistemic state e. A world determines truth values for the ground atoms after any sequence of actions. An epistemic state is defined by a set of worlds, as in possible-world semantics.

More precisely, let $\mathcal{Z}$ be the set of all finite sequences of elements of $\mathcal{R}$ including $\langle\,\rangle$, the empty sequence. $\mathcal{Z}$ should be understood as the set of all finite sequences of actions. Then

– a world $w \in W$ is any function from $\mathcal{G}$ (the set of ground atoms) and $\mathcal{Z}$ to $\{0, 1\}$.
– an epistemic state $e \subseteq W$ is any set of worlds.

To interpret formulas with free variables, we proceed as follows. First-order variables are handled substitutionally using the rigid terms $\mathcal{R}$. To handle the quantification over second-order variables, we use second-order *variable maps* defined as follows:

> The *second-order ground atoms* are formulas of the form $V(t_1, \ldots, t_k)$ where V is a second-order variable and all of the t_i are in $\mathcal{R}$. A *variable map* u is a function from second-order ground atoms to $\{0, 1\}$.

[4] In the situation calculus, these correspond to formulas that are uniform in some situation term.

Let u and u' be variable maps, and let V be a second-order variable; we write $u' \sim_V u$ to mean that u and u' agree except perhaps on the assignments involving V.

Finally, to interpret what is known after a sequence of actions has taken place, we define $w' \simeq_z w$ (read: w' agrees with w on the sensing throughout action sequence z) inductively by the following:

1. $w' \simeq_{\langle\rangle} w$ for all w';
2. $w' \simeq_{z \cdot t} w$ iff $w' \simeq_z w$,
   $w'[\mathit{Poss}(t), z] = 1$ and $w'[\mathit{SF}(t), z] = w[\mathit{SF}(t), z]$.

Note that $\simeq_z$ is not quite an equivalence relation because of the use of *Poss* here. This is because we are insisting that the agent comes to believe that *Poss* was true after performing an action, even in those situations where the action was not executable in reality.[5]

Putting all these together, we now turn to the semantic definitions for sentences of $\mathcal{ES}_0$. Given an epistemic state $e \subseteq W$, a world $w \in W$, an action sequence $z \in \mathcal{Z}$, and a second-order variable map u, we have:

1. $e, w, z, u \models F(t_1, \ldots, t_k)$ iff $w[F(t_1, \ldots, t_k), z] = 1$;
2. $e, w, z, u \models V(t_1, \ldots, t_k)$ iff $u[V(t_1, \ldots, t_k)] = 1$;
3. $e, w, z, u \models (t_1 = t_2)$ iff t_1 and t_2 are identical;
4. $e, w, z, u \models [t]\alpha$ iff $e, w, z \cdot t, u \models \alpha$;
5. $e, w, z, u \models (\alpha \wedge \beta)$ iff
   $e, w, z, u \models \alpha$ and $e, w, z, u \models \beta$;
6. $e, w, z, u \models \neg\alpha$ iff $e, w, z, u \not\models \alpha$;
7. $e, w, z, u \models \forall x.\, \alpha$ iff $e, w, z, u \models \alpha^x_t$, for all $t \in \mathcal{R}$;
8. $e, w, z, u \models \forall V.\, \alpha$ iff
   $e, w, z, u' \models \alpha$, for all $u' \sim_V u$;
9. $e, w, z, u \models \Box\alpha$ iff $e, w, z \cdot z', u \models \alpha$, for all $z' \in \mathcal{Z}$;

To define the meaning of the epistemic operators, we need the following definition:

Definition 4. *Let w be a world and e a set of worlds, and z a sequence of actions. Then*

1. *w_z is a world such that $w_z[p, z'] = w[p, z \cdot z']$ for all ground atoms p and all z';*
2. *$e^w_z = \{w'_z \mid w' \in e$ and $w' \simeq_z w\}$.*

Note that w_z is exactly like w after the actions z have occurred. So in a sense, w_z can be thought of as the progression of w wrt z. e^w_z then contains all those worlds of e which are progressed wrt z and which are compatible with (the real) world w in terms of the sensing results and where all the actions in z are executable. Note that when z is empty, $e^w_z = e$.

10. $e, w, z, u \models \boldsymbol{K}\alpha$ iff for all $w' \in e^w_z$, $e^w_z, w', \langle\rangle, u \models \alpha$;
11. $e, w, z, u \models \boldsymbol{O}\alpha$ iff for all w', $w' \in e^w_z$ iff $e^w_z, w', \langle\rangle, u \models \alpha$.

[5] An alternate account that would state that the agent learns the true value of *Poss* (analogous to *SF*) is a bit more cumbersome, but would allow $\simeq_z$ to be a full equivalence relation.

In other words, knowing α in e and w after actions z means that α is true in all the progressed worlds of e which are compatible with w. $\boldsymbol{O}\alpha$ is quite similar except for the "iff," whose effect is that e_z^w must contain every world which satisfies α.

$\boldsymbol{B}$ and $\boldsymbol{\Omega}$ are meant to capture what the agent believes in addition by applying defaults. Having more beliefs (as a result of defaults) is modeled by considering a subset of the worlds in e_z^w. For that purpose, we introduce a function δ which maps each set of worlds into a subset. In particular, we require that $\delta(e_z^w) \subseteq e_z^w$. As δ is now part of the model (just like w and e) we add it to the L.H.S. of the satisfaction relation with the understanding that the previous rules are retrofitted with δ as well. Then we have:

12. $e, w, z, u, \delta \models \boldsymbol{B}\alpha$ iff for all $w' \in \delta(e_z^w)$, $e_z^w, w', \langle\rangle, u, \delta \models \alpha$;
13. $e, w, z, u, \delta \models \boldsymbol{\Omega}\alpha$ iff for all $w' \in e_z^w$, $w' \in \delta(e_z^w)$ iff $e_z^w, w', \langle\rangle, u, \delta \models \alpha$.

Note that the only difference between $\boldsymbol{K}$ and $\boldsymbol{B}$ is that the latter considers $\delta(e_z^w)$ instead of e_z^w. Likewise, the definition of $\boldsymbol{\Omega}$ is similar to that of $\boldsymbol{O}$. The role of $\boldsymbol{\Omega}$ is to constrain δ to produce a special subset of e_z^w. Roughly, the effect of the definition of $\boldsymbol{\Omega}\alpha$ is that one starts with whatever facts are believed (represented by e_z^w) and then settles on a largest subset of e_z^w such that α (representing the defaults) is also believed.

We say that a sentence in $\mathcal{ES}_0$ is true at a given e, w, and δ (written $e, w, \delta \models \alpha$) if $e, w, \langle\,\rangle, u, \delta \models \alpha$ for any second-order variable map u. If Σ is a set of sentences and α is a sentence, we write $\Sigma \models \alpha$ (read: Σ logically entails α) to mean that for every e, w, and δ, if $e, w, \delta \models \alpha'$ for every $\alpha' \in \Sigma$, then $e, w, \delta \models \alpha$. Finally, we write $\models \alpha$ (read: α is valid) to mean $\{\} \models \alpha$.

2.3 Some Properties

We begin with some simple properties relating the various epistemic operators. Note the leading $\Box$ operator in all cases, which means that these properties hold after any number of actions have occurred.

Proposition 1

1. $\models \Box(\boldsymbol{K}\alpha \supset \boldsymbol{B}\alpha)$
2. $\models \Box(\boldsymbol{O}\alpha \supset \boldsymbol{K}\alpha)$
3. $\models \Box(\boldsymbol{\Omega}\alpha \supset \boldsymbol{B}\alpha)$

Proof:

1. Let $e, w, z, u, \delta \models \boldsymbol{K}\alpha$. Then for all $w' \in e_z^w$, $e_z^w, w', \langle\,\rangle, u, \delta \models \alpha$. Since $\delta(e_z^w) \subseteq e_z^w$ by the definition of δ, we have that for all $w' \in \delta(e_z^w)$, $e_z^w, w', \langle\,\rangle, u, \delta \models \alpha$. Hence $e, w, z, u, \delta \models \boldsymbol{B}\alpha$.
2. Let $e, w, z, u, \delta \models \boldsymbol{O}\alpha$. Then for all $w' \in e_z^w$, $e_z^w, w', \langle\,\rangle, u, \delta \models \alpha$ by the definition of $\boldsymbol{O}$, and hence $e, w, z, u, \delta \models \boldsymbol{K}\alpha$.
3. The argument is analogous.

It is also not hard to see that $\boldsymbol{K}$ and $\boldsymbol{B}$ satisfy the usual $K45$ axioms of modal logic (Hughes and Cresswell, 1968), again, after any number of actions. For example, we get full introspection for both $\boldsymbol{K}$ and $\boldsymbol{B}$. Moreover, introspection is mutual.

Proposition 2. *Let* $\boldsymbol{L}, \boldsymbol{M} \in \{\boldsymbol{K}, \boldsymbol{B}\}$.

1. $\models \Box(\boldsymbol{L}\alpha \supset \boldsymbol{M}\boldsymbol{L}\alpha)$
2. $\models \Box(\neg\boldsymbol{L}\alpha \supset \boldsymbol{M}\neg\boldsymbol{L}\alpha)$

Proof: As the arguments are all similar, we only prove two cases:

1. $\models \Box(\boldsymbol{B}\alpha \supset \boldsymbol{K}\boldsymbol{B}\alpha)$.
   Let $e, w, z, u, \delta \models \boldsymbol{B}\alpha$. Then for all $w' \in \delta(e^w_z)$, $e^w_z, w', \langle\,\rangle, u, \delta \models \alpha$. Now let $w^* \in e^w_z$. We need to show that $e^w_z, w^*, \langle\,\rangle, u, \delta \models \boldsymbol{B}\alpha$, that is, for all $w'' \in \delta((e^w_z)_{\langle\,\rangle})$, $\delta((e^w_z)_{\langle\,\rangle}), w'', \langle\,\rangle, u, \delta \models \alpha$. Since $(e^w_z)_{\langle\,\rangle} = e^w_z$, this follows immediately.
2. $\models \Box(\neg\boldsymbol{K}\alpha \supset \boldsymbol{B}\neg\boldsymbol{K}\alpha)$.
   Let $e, w, z, u, \delta \models \neg\boldsymbol{K}\alpha$, that is, for some $w' \in e^w_z$, $e^w_z, w', \langle\,\rangle, u, \delta \not\models \alpha$. Since $(e^w_z)_{\langle\,\rangle} = e^w_z$, we have that for some $w' \in (e^w_z)_{\langle\,\rangle}$, $(e^w_z)_{\langle\,\rangle}, w', \langle\,\rangle, u, \delta \not\models \alpha$. Then for any $w^* \in \delta(e^w_z)$, $e^w_z, w^*, \langle\,\rangle, u, \delta \models \neg\boldsymbol{K}\alpha$. Hence $e, w, z, u, \delta \models \boldsymbol{B}\neg\boldsymbol{K}\alpha$.

We end our brief discussion of the properties of $\mathcal{ES}_{\text{O}}$ with some useful lemmas about the progression of worlds and epistemic states. The first says that first progressing a world w by z and then progressing the result w_z by z' is the same as progressing w directly by $z \cdot z'$.

Lemma 1. $(w_z)_{z'} = w_{z\cdot z'}$.

Proof: Suppose p is any primitive proposition and z^* a sequence of actions. Then, by Definition 4, $(w_z)_{z'}[p, z^*] = w_z[p, z' \cdot z^*] = w[p, z \cdot z' \cdot z^*] = w_{z\cdot z'}[p, z^*]$. ∎

Lemma 2. $w' \simeq_{z\cdot z'} w$ *iff* $w' \simeq_z w$ *and* $w'_z \simeq_{z'} w_z$.

Proof: The proof is by induction on z'. The lemma holds trivially if $z' = \langle\,\rangle$, since $z \cdot z' = z$ and $w'_z \simeq_{z'} w$ holds vacuously in this case. Suppose the lemma holds for some z' and now consider $z' \cdot t$ for some action t. Let $w' \simeq_{z\cdot z'\cdot t} w$. Since $w' \simeq_{z\cdot z'} w$ holds by assumption and the definition of $\simeq_z$, by induction, $w' \simeq_z w$ and $w'_z \simeq_{z'} w_z$. Also $w'[\mathit{Poss}(n), z \cdot z'] = 1$ and $w'[\mathit{SF}(t), z \cdot z'] = w[\mathit{SF}(t), z \cdot z']$ and, hence, $w'_z[\mathit{Poss}(t), z'] = 1$ and $w'_z[\mathit{SF}(t), z'] = w_z[\mathit{SF}(t), z']$. Therefore, $w'_z \simeq_{z'\cdot t} w_z$. Conversely, let $w' \simeq_z w$ and $w'_z \simeq_{z'\cdot t} w_z$. Then $w' \simeq_{z\cdot z'} w$ holds by induction. Also, $w'[\mathit{Poss}(n), z \cdot z'] = 1$ and $w'[\mathit{SF}(t), z \cdot z'] = w[\mathit{SF}(t), z \cdot z']$ follows from $w'_z \simeq_{z'\cdot t} w_z$. Hence $w' \simeq_{z\cdot z'\cdot t} w$. ∎

Lemma 3. $(e^w_z)^{w_z}_{z'} = e^w_{z\cdot z'}$.

Proof: By definition, $(e^w_z)^{w_z}_{z'} = \{w'_{z'} \mid w' \in e^w_z \text{ and } w' \simeq_{z'} w_z\}$, which is the same as $\{w'_{z'} \mid \text{for some } w^* \in e, w' = w^*_z,\ w^* \simeq_z w, \text{ and } w' \simeq_{z'} w_z\}$. Substituting w^*_z for w', this set is the same as $\{(w^*_z)_{z'} \mid w^* \in e, w^* \simeq_z w, \text{ and } w' \simeq_{z'} w_z\}$, which, using the previous two lemmas, simplifies to $\{w^*_{z\cdot z'} \mid w^* \in e, w^* \simeq_{z\cdot z'} w\}$, which is $e^w_{z\cdot z'}$. ∎

Lemma 4. *If* α *contains no* $\boldsymbol{O}$ *and* $\boldsymbol{\Omega}$, *then* $e, w, z \cdot z', u, \delta \models \alpha$ *iff* $e^w_z, w_z, z', u, \delta \models \alpha$.

Proof: The proof is by induction on α. The base case holds by the construction of w_z. The cases $\neg$, $\wedge$, and $\forall$ clearly hold by induction. $e, w, z \cdot z', u, \delta \models [t]\alpha$ iff $e, w, z \cdot z' \cdot t, u, \delta \models \alpha$ iff (by induction) $e^w_z, w_z, z' \cdot t, u, \delta \models \alpha$ iff $e^w_z, w_z, z', u, \delta \models [t]\alpha$. $e, w, z \cdot z', u, \delta \models \Box\alpha$ iff $e, w, z \cdot z' \cdot z'', u, \delta \models \alpha$ for all $z'' \in \mathcal{Z}$ iff (by ind.) $e^w_z, w_z, z' \cdot z'', u, \delta \models \alpha$ for all $z'' \in \mathcal{Z}$ iff $e^w_z, w_z, z', u, \delta \models \Box\alpha$.

$e, w, z \cdot z', u, \delta \models \boldsymbol{K}\alpha$ iff for all $w' \in e^w_{z \cdot z'}$, $e^w_{z \cdot z'}, w', \langle\rangle, u, \delta \models \alpha$ iff (by Lemma 3) for all $w' \in (e^w_z)^{w_z}_{z'}$, $(e^w_z)^{w_z}_{z'}, w', \langle\rangle, u, \delta \models \alpha$ iff $e^w_z, w_z, z', u, \delta \models \boldsymbol{K}\alpha$.

$e, w, z \cdot z', u, \delta \models \boldsymbol{B}\alpha$ iff for all $w' \in \delta(e^w_{z \cdot z'})$, $e^w_{z \cdot z'}, w', \langle\rangle, u, \delta \models \alpha$ iff for all $w' \in \delta((e^w_z)^{w_z}_{z'})$, $(e^w_z)^{w_z}_{z'}, w', \langle\rangle, u, \delta \models \alpha$ iff $e^w_z, w_z, z', u, \delta \models \boldsymbol{B}\alpha$. ∎

3 The Semantics of Progression and Defaults

3.1 Basic Action Theories

Let us now consider the equivalent of basic action theories of the situation calculus. Since in our logic there is no explicit notion of situations, our basic action theories do not require foundational axioms like Reiter's (2001) second-order induction axiom for situations. The treatment of defaults is deferred to Section 3.4.

Definition 5. *Given a set of fluents $\mathcal{F}$, a set $\Sigma \subseteq \mathcal{ES}_0$ of sentences is called a basic action theory over $\mathcal{F}$ iff*
$\Sigma = \Sigma_0 \cup \Sigma_{\text{pre}} \cup \Sigma_{\text{post}} \cup \Sigma_{\text{sense}}$ *where*

1. *Σ_0 is any set of fluent sentences;*
2. *Σ_{pre} is a singleton sentence of the form $\Box Poss(a) \equiv \pi$, where π is a fluent formula;*[6]
3. *Σ_{post} is a set of sentences of the form $\Box[a]F(\boldsymbol{v}) \equiv \gamma_F$, one for each relational fluent F in $\mathcal{F}$, respectively, and where the γ_F are fluent formulas.*[7]
4. *Σ_{sense} is a sentence exactly parallel to the one for Poss of the form $\Box SF(a) \equiv \varphi$, where φ is a fluent formula.*

The idea here is that Σ_0 expresses what is true initially (in the initial situation), Σ_{pre} is one large precondition axiom, and Σ_{post} is a set of successor state axioms, one per fluent in $\mathcal{F}$, which incorporate the solution to the frame problem proposed by Reiter (1991). Σ_{sense} characterizes the sensing results of actions. For actions like *drop*(o), which do not return any useful sensing information, *SF* can be defined to be vacuously true (see below for an example).

We will usually require that Σ_{pre}, Σ_{post}, and Σ_{sense} be first-order. However, Σ_0 may contain second-order sentences. As we will see, this is inescapable if we want to capture progression correctly. In the following, we assume that Σ (and hence $\mathcal{F}$) is finite and we will freely use Σ or its subsets as part of sentences with the understanding that we mean the conjunction of the sentences contained in the set.

[6] We assume that all free variables are implicitly universally quantified and that $\Box$ has lower syntactic precedence than the logical connectives, so that $\Box Poss(a) \equiv \pi$ stands for the sentence $\forall a.\Box(Poss(a) \equiv \pi)$.

[7] The $[t]$ construct has higher precedence than the logical connectives. So $\Box[a]F(\boldsymbol{x}) \equiv \gamma_F$ abbreviates the sentence $\forall a, \boldsymbol{x}.\Box([a]F(\boldsymbol{x}) \equiv \gamma_F)$.

3.2 Progression = Only-Knowing After an Action

Let us now turn to the first main result of this paper. The question we want to answer is this: suppose an agent is given a basic action theory as its initial knowledge base; how do we characterize the agent's knowledge after an action is performed? As hinted in the introduction, only-knowing will give us the answer.

In the following, for a given basic action theory Σ, we sometimes write ϕ for Σ_0 and $\Box\beta$ for the rest of the action theory $\Sigma_{\text{pre}} \cup \Sigma_{\text{post}} \cup \Sigma_{\text{sense}}$. We assume that π and φ refer to the right-hand sides of the definitions of *Poss* and *SF* in Σ, and γ_F is the right-hand side of the successor state axiom for fluent F. Also, let $\boldsymbol{F}$ consist of all the fluent predicate symbols in Σ, and let $\boldsymbol{P}$ be corresponding second-order variables, where each P_i has the same arity as F_i. Then $\alpha^{\boldsymbol{F}}_{\boldsymbol{P}}$ denotes the formula α with every occurrence of F_i replaced by P_i.

The following result characterizes in general terms all that is known after performing an action:

Theorem 1. *Let $\Sigma = \phi \wedge \Box\beta$ be a basic action theory and t an action term. Then*

$$\begin{aligned}\models \boldsymbol{O}(\phi \wedge \Box\beta) \supset &\\ &(SF(t) \supset [t]\boldsymbol{O}(\Psi \wedge \Box\beta)) \wedge\\ &(\neg SF(t) \supset [t]\boldsymbol{O}(\Psi' \wedge \Box\beta)),\end{aligned}$$

where $\Psi = \exists \boldsymbol{P}[(\phi \wedge \pi^a_t \wedge \varphi^a_t)^{\boldsymbol{F}}_{\boldsymbol{P}} \wedge \bigwedge \forall \boldsymbol{x}.F(\boldsymbol{x}) \equiv \gamma_F{}^{a}_{t}{}^{\boldsymbol{F}}_{\boldsymbol{P}}]$ *and*
$\Psi' = \exists \boldsymbol{P}[(\phi \wedge \pi^a_t \wedge \neg\varphi^a_t)^{\boldsymbol{F}}_{\boldsymbol{P}} \wedge \bigwedge \forall \boldsymbol{x}.F(\boldsymbol{x}) \equiv \gamma_F{}^{a}_{t}{}^{\boldsymbol{F}}_{\boldsymbol{P}}]$.

What the theorem says is that if all the agent knows initially is a basic action theory, then after doing action t all the agent knows is another basic action theory, where the dynamic part ($\Box\beta$) remains the same and the initial database ϕ is replaced by Ψ or Ψ', depending on the outcome of the sensing. Note that the two sentences differ only in one place, φ^a_t vs. $\neg\varphi^a_t$. Roughly, Ψ and Ψ' specify how the truth value of each fluent F in $\mathcal{F}$ is determined by what was true previously (ϕ), taking into account that the action was possible (π^a_t) and that the sensing result was either true (φ^a_t) or false ($\neg\varphi^a_t$). Since after performing an action, the agent again only-knows a basic action theory, we can take this as its new initial theory and the process can iterate. We remark that our notion of progression is very closely related to progression as defined by (Lin and Reiter, 1997), but extends it to handle sensing actions. Note that, while Lin and Reiter need to include the unique names axioms for actions in the progression, we do not, as these are built into the logic.

The proof of the theorem requires two lemmas. When formulas are objective or don't mention $\boldsymbol{B}$ and $\boldsymbol{\Omega}$, their satisfaction is independent of any e or δ. Hence we will leave those out in the following wherever possible.

Lemma 5. *Let $w', \langle\rangle, u \models \phi \wedge \Box\beta$ and $w' \simeq_t w$. Then $w'_t, \langle\rangle, u \models \Psi \wedge \Box\beta$.*

Proof: Since $w', \langle\rangle, u \models \Box\beta$, we have $w', \langle\rangle, u \models [t]\Box\beta$ and thus $w', t, u \models \Box\beta$. By Lemma 4, $w'_t, \langle\rangle, u \models \Box\beta$. We are left to show that $w'_t, \langle\rangle, u \models \Psi$. Let u' be a variable map such that $u' \sim_{\boldsymbol{P}} u$ and $u'[P_i(t_1, \ldots, t_k)] = w'[F_i(t_1, \ldots, t_k), \langle\rangle]$. Note also that $w', \langle\rangle, u \models \phi$ and $w', \langle\rangle, u \models \pi^a_t \wedge \varphi^a_t$ because $w' \simeq_t w$. It is easy to show (by induction) that $w', t, u' \models (\phi \wedge \pi_t^a \wedge \varphi_t^a)_{\boldsymbol{P}}^{\boldsymbol{F}}$ and, by Lemma 4, $w'_t, \langle\rangle, u' \models (\phi \wedge \pi_t^a \wedge \varphi_t^a)_{\boldsymbol{P}}^{\boldsymbol{F}}$. Similarly, we can show that $w'_t, \langle\rangle, u' \models \forall \boldsymbol{x}.F(\boldsymbol{x}) \equiv \gamma_F{}_t^a{}_{\boldsymbol{P}}^{\boldsymbol{F}}$ for all $F \in \mathcal{F}$. Hence $w'_t, \langle\rangle, u \models \Psi$. ▮

Lemma 6. *Let $w^*, \langle\rangle, u \models \Psi \wedge \Box\beta$. Then there exists a w^{**} such that $w^* = w_t^{**}$, $w^{**} \simeq_t w$, and $w^{**}, \langle\rangle, u \models \phi \wedge \Box\beta$.*

Proof: Let $w^*, \langle\rangle, u \models \Psi \wedge \Box\beta$. Then for some $u' \sim_{\boldsymbol{P}} u$, $w^*, \langle\rangle, u' \models (\phi \wedge \pi_t^a \wedge \varphi_t^a)_{\boldsymbol{P}}^{\boldsymbol{F}} \wedge \bigwedge \forall \boldsymbol{x}.F(\boldsymbol{x}) \equiv \gamma_F{}_t^a{}_{\boldsymbol{P}}^{\boldsymbol{F}}$. Construct w^{**} from u' inductively:

1. $w^{**}[F_i(\boldsymbol{t}), \langle\rangle] = u'[P_i(\boldsymbol{t})]$ for all fluents in $\phi \wedge \Box\beta$;
2. $w^{**}[Poss(t), \langle\rangle] = w^{**}[SF(t), \langle\rangle] = 1$;
3. $w^{**}[F(\boldsymbol{t}), \langle\rangle]$ is arbitrary for all other fluents F.
   Now suppose $w^{**}[p, z]$ is defined for z.
4. $w^{**}[F(\boldsymbol{t}), z \cdot s] = 1$ iff $w^{**}, z, u \models \gamma_F{}_{\boldsymbol{t}}^{\boldsymbol{x}}{}_s^a$ for $F \in \mathcal{F}$;
5. $w^{**}[Poss(s'), z \cdot s] = 1$ iff $w^{**}, z, u \models \pi_{s'}^a$;
6. $w^{**}[SF(s'), z \cdot s] = 1$ iff $w^{**}, z, u \models \varphi_{s'}^a$;
   For all other fluents F:
7. if $z \cdot s = t \cdot z'$ then $w^{**}[F(\boldsymbol{t}), z \cdot s] = w^*[F(\boldsymbol{t}), z']$;
8. if $z \cdot s \neq t \cdot z'$ then $w^{**}[F(\boldsymbol{t}), z \cdot s]$ is arbitrary.

Since $w^*, \langle\rangle, u' \models \phi_{\boldsymbol{P}}^{\boldsymbol{F}}$ we obtain, by construction, $w^{**}, \langle\rangle, u \models \phi$. Similarly we have $w^{**}, \langle\rangle, u \models \pi_t^a \wedge \varphi_t^a$, that is, $w^{**} \simeq_t w$. Also, by construction, $w^* = w_t^{**}$. Finally, it is not difficult to show that $w^{**}, \langle\rangle, u \models \Box\beta$, and we are done. ▮

We are now ready for the proof of the main theorem.

Proof: Here we consider only the case where *SF*(t) holds. The other is completely symmetric. Let $e, w, \langle\rangle, u \models \boldsymbol{O}(\phi \wedge \Box\beta) \wedge SF(t)$. We need to show that $e, w, t, u \models \boldsymbol{O}(\Psi \wedge \Box\beta)$, that is, for all w^*, $w^* \in e_t^w$ iff $w^*, \langle\rangle, u \models \Psi \wedge \Box\beta$.

To show the only-if direction, let $w^* \in e_t^w$, that is, $w^* = w'_t$ for some $w' \in e$ and $w' \simeq_t w$ by the definition of e_t^w. Since $w' \models \Psi \wedge \Box\beta$ by assumption, $w'_t \models \alpha$ by Lemma 5.

Conversely, let $w^*, \langle\rangle, u \models \Psi \wedge \Box\beta$. (We can ignore the epistemic part because the sentence is objective.) By Lemma 6, there is a w^{**} with $w^* = w_t^{**}$ and $w^{**}, \langle\rangle, u \models \phi \wedge \Box\beta$ and $w^{**} \simeq_t w$. Hence, by assumption, $w^{**} \in e$ and thus $w_t^{**} \in e_t^w$, that is, $w^* \in e_t^w$. ▮

We mentioned above that after an action, the resulting knowledge base can be taken as the new initial knowledge base, and the progression can iterate. The following theorem shows that this view is justified in that the entailments about the future remain the same when we substitute what is known about the world initially by its progression. Here we only consider the case where *SF*(t) is true.

Theorem 2. $\models \boldsymbol{O}(\phi \wedge \Box\beta) \wedge SF(t) \supset [t]\boldsymbol{K}(\alpha)$ *iff* $\models \boldsymbol{O}(\Psi \wedge \Box\beta) \supset \boldsymbol{K}(\alpha)$.

In English (roughly): It follows from your initial knowledge base that you will know α after doing action t iff knowing α follows from your progressed knowledge base. The proof uses techniques very similar to the previous theorem and lemmas.

3.3 Comparison with Lin and Reiter

Those familiar with the work by Lin and Reiter on progression in the situation calculus (Lin and Reiter, 1997) will have noticed that our notion of progression is very close to theirs. To see the similarities and differences, let us briefly review their definition, which we call LR-progression.

Lin and Reiter give a model-theoretic definition of progression and show that, when the initial database is finite, it is always representable using a second-order formula similar to ours. As the two views are equivalent for finite initial databases, we will not introduce the model-theoretic definition but simply use Lin and Reiter's syntactic representation instead, phrased in the language of $\mathcal{ES}_0$ (see (Lakemeyer and Levesque, 2005) for how to translate formulas $\mathcal{ES}_0$ into formulas of the classical situation calculus).

Definition 6 (LR-Progression)
Let Σ be a basic action theory over fluents $\boldsymbol{F}$, and let t be a ground action term. An LR-progression is any set of sentences which is logically equivalent to Σ_{una}, the unique name axioms for actions, conjoined with the following sentence (as usual, let ϕ be Σ_0):

$$\exists \boldsymbol{P}.\ \phi^{\boldsymbol{F}}_{\boldsymbol{P}} \ \wedge\ [\pi^{a}_{t}{}^{\boldsymbol{F}}_{\boldsymbol{P}} \ \supset\ \bigwedge \forall \boldsymbol{x}(F(\boldsymbol{x})\ \equiv\ \gamma_{F}{}^{a}_{t}{}^{\boldsymbol{F}}_{\boldsymbol{P}})].$$

The idea here is that an LR-progression together with the rest of the basic action theory entails the same sentences about the future after t has been performed as the original basic action theory, similar to our Theorem 2. Despite the apparent similarities, there are also differences:

1. Lin and Reiter require the unique name axioms for actions. We do not, essentially because the unique name assumption is built into the logic.
2. *Poss* is handled differently. In part, this has to do with an older version of successor state axioms used by Lin and Reiter. These had the form

$$\Box Poss(a) \supset [a]F(\boldsymbol{x}) \equiv \gamma_F.$$

   Hence it was necessary to replace $Poss(a)$ by its definition in the progression. However, unlike us, Lin and Reiter do not conclude after executing an action that it was possible. It is easy to see though that, in case ϕ logically entails π^a_t, our handling of *Poss* coincides with Lin and Reiter's.
3. There is no notion of sensing, which is not surprising as Lin and Reiter do not consider knowledge and their basic action theories do not contain Σ_{sense}. Here, we believe, our proposal provides new insights, both in terms of what progression means in an epistemic setting and with regards to actions with non-trivial sensing results.

Lin and Reiter gave examples of classes of basic action theories where the result of progression is first-order representable. By adapting their proofs to $\mathcal{ES}_0$, it is not difficult to show that those results carry over to progression based on only-knowing as well. We can also show that, in the case of pure sensing actions, which themselves do not change any fluents, first-order representability is also guaranteed:

Definition 7. *Given a basic action theory Σ, a ground action t is called a pure sensing action if for every fluent in $\mathcal{F}$, $\Sigma_{\text{post}} \models \Box\forall \boldsymbol{x}.\ F(\boldsymbol{x}) \equiv [t]F(\boldsymbol{x})$.*

Theorem 3. *Let t be a pure sensing action. Then*
$\models \boldsymbol{O}(\phi \wedge \Box\beta) \wedge SF(t) \supset [t]\boldsymbol{O}(\phi \wedge \pi^a_t \wedge \varphi^a_t \wedge \Box\beta)$.

Comparison with the Old Only-Knowing. Before turning to defaults, let us briefly compare our version of $\boldsymbol{O}$ with the one in (Lakemeyer and Levesque, 2004). For that we add a new operator $\boldsymbol{O'}$ to the language with the following semantics:

11′. $e, w, z, u \models \boldsymbol{O'}\alpha$ iff
for all w' such that $w' \simeq_z w$, $w' \in e$ iff $e, w', z, u \models \alpha$.

Technically there are two main differences between this and our new definition of $\boldsymbol{O}$. For one $\boldsymbol{O'}$ only considers worlds w' that are compatible with w via $\simeq_z$. For another, the truth of α is considered wrt the original e and z and not wrt the progressed worlds in e_z^w.

In the initial situation, before any actions, the two definitions actually coincide. In other words, we have

Proposition 3. $\models \boldsymbol{O}\alpha \equiv \boldsymbol{O'}\alpha$.

The proof is trivial because initially $z = \langle\,\rangle$ and $e_z^w = e$ and $w^* \simeq_z w$ holds for all worlds. Things diverge significantly once an action has been performed, and this is best illustrated using a basic action theory. Recall that in our example we had

$$\models \Sigma \wedge \boldsymbol{O}(\phi \wedge \Box\beta) \supset [drop(x)]\boldsymbol{O}(\Psi \wedge \Box\beta),$$

where $\phi = \mathit{Fragile}(o) \wedge \neg \mathit{Broken}(o)$ and Ψ is equivalent to $(\mathit{Fragile}(o) \wedge \mathit{Broken}(o))$.

Now let $e, w, \langle\,\rangle, u \models \Sigma \wedge \boldsymbol{O'}(\phi \wedge \Box\beta)$ and let $t = drop(o)$. It is not hard to show that there is a world w^* such that $w^* \simeq_t w$ such that $w^*, \langle\,\rangle, u \models \mathit{Broken}(o)$ and $w^*, t, u \models \Psi \wedge \Box\beta$. Such a world cannot be in e as initially o is known not to be broken. Therefore, we have that $e, w, \langle\,\rangle, u \not\models [drop(x)]\boldsymbol{O'}(\Psi \wedge \Box\beta)$. The trouble is, roughly, that the complete worlds of e are kept around in the definition of $\boldsymbol{O'}$, which forces the agent to remember what was true in the past, while the new $\boldsymbol{O}$ allows the agent to forget the past, as with Lin-Reiter progression.

3.4 Defaults for Basic Action Theories

Here we restrict ourselves to static defaults like “birds normally fly.” In an autoepistemic setting (Moore, 1985; Levesque, 1990), these have the following form:

$$\forall \boldsymbol{x}. \boldsymbol{B}\alpha \wedge \neg \boldsymbol{B}\neg\beta \supset \gamma,$$

which can be read as “if α is believed and β is consistent with what is believed then assume γ.” Here the assumption is that α, β, and γ are static objective formulas.

Let Σ_{def} be the conjunction of all defaults of the above form held by an agent. For a given basic action theory Σ, as defined in Section 3.1, the idea is to apply the same defaults to what is known about the current situation after any number of actions have occurred, that is, for the purpose of default reasoning, we assume that $\Box\Omega\Sigma_{\text{def}}$ holds. We will now relate what is believed in the presence of defaults to the *stable expansions* of Moore’s autoepistemic logic (Moore, 1985).

Definition 8 (Stable expanion)
Let α, β, γ be first-order static belief sentences and E a set of such sentences.

E is called a stable expansion of α iff for all γ, $\gamma \in E$ iff γ is a first-order consequence of $\{\alpha\} \cup \{\boldsymbol{B}\beta \mid \beta \in E\} \cup \{\neg\boldsymbol{B}\beta \mid \beta \notin E\}$.

In words, a stable expansion of α consists of those sentences which can be obtained from α by logical reasoning and introspection alone. For example, the default

$$\forall x \neg \boldsymbol{B} \neg \mathit{Fragile}(x) \supset \mathit{Fragile}(x)$$

has exactly one stable expansion and its objective sentences are precisely those which follow from $\forall x.\mathit{Fragile}(x)$, that is, the expansion correctly captures the default conclusions sanctioned by the default rule. We remark that, in general, there may be more than one stable expansion, or none at all.

The following theorem shows the connection between what is believed after an action has occurred (where *SF* returns true) and stable expansions.

Theorem 4. *Let t be a ground action and $\Sigma = \phi \wedge \Box\beta$ a basic action theory such that $\models \boldsymbol{O}\Sigma \wedge \mathit{SF}(t) \supset [t]\boldsymbol{O}(\psi \wedge \Box\beta)$, where ψ is first order. Then for any static belief sentence γ,*

$$\models \boldsymbol{O}\Sigma \wedge \mathit{SF}(t) \wedge \Box\Omega\Sigma_{\text{def}} \supset [t]\boldsymbol{B}\gamma \quad \textit{iff } \gamma \textit{ is in every stable expansion of } \psi \wedge \Sigma_{\text{def}}.$$

We remark that an analogous theorem holds also for $\neg \mathit{SF}(t)$ and that the restriction to a first-order progression is made only because stable expansions were originally defined only for first-order theories.

3.5 An Example

To illustrate progression, let us consider the example of the introduction with two fluents *Broken* and *Fragile*, actions $\mathit{drop}(x)$, $\mathit{repair}(x)$, and $\mathit{senseF}(x)$ (for sensing whether x is fragile). First, we let the basic action theory Σ consist of the following axioms:

- $\Sigma_0 = \{\mathit{Fragile}(o), \neg \mathit{Broken}(o)\}$;
- $\Sigma_{\text{pre}} = \{\Box \mathit{Poss}(a) \equiv \mathit{true}\}$ (for simplicity);
- $\Sigma_{\text{post}} = \{\mathit{SSA}_{BF}\}$ (from the introduction);
- $\Sigma_{\text{sense}} = \{\Box \mathit{SF}(a) \equiv \exists x.a = \mathit{drop}(x) \wedge \mathit{true} \vee$
  $a = \mathit{repair}(x) \wedge \mathit{true} \ \vee \ a = \mathit{senseF}(x) \wedge \mathit{Fragile}(x)\}$.

As before, let $\Box\beta$ be $\Sigma_{\text{pre}} \cup \Sigma_{\text{post}} \cup \Sigma_{\text{sense}}$. Then we have

$$\models \Sigma \wedge \boldsymbol{O}(\Sigma_0 \wedge \Box\beta) \supset [\mathit{drop}(o)]\boldsymbol{O}(\Psi \wedge \Box\beta),$$

where $\Psi = \exists P, P'.[\neg P(o) \wedge P'(o) \wedge$
$\exists x.\mathit{drop}(o) = \mathit{drop}(x) \wedge \mathit{true} \vee$
$\mathit{drop}(o) = \mathit{repair}(x) \wedge \mathit{true} \vee$
$\mathit{drop}(o) = \mathit{senseF}(x) \wedge P'(x) \wedge$
$\forall x.\mathit{Broken}(x) \equiv \mathit{drop}(o) = \mathit{drop}(x) \wedge P'(x) \vee$
$P(x) \wedge \mathit{drop}(o) \neq \mathit{repair}(x) \wedge$
$\forall x.\mathit{Fragile}(x) \equiv P'(x)]$.

Using the fact that all actions are distinct, it is not difficult to see that Ψ can be simplified to

$(\mathit{Fragile}(o) \wedge \mathit{Broken}(o))$.

In other words, after dropping o, the agent's knowledge base is as before, except that o is now known to be broken.

To see how defaults work, we now let Σ be as before except that $\Sigma_0 = \{\neg \mathit{Broken}(o)\}$ and let $\Sigma' = \Sigma \cup \{\neg \mathit{Fragile}(o)\}$. Let $\Sigma_{\mathrm{def}} = \{\forall x. \neg \boldsymbol{B} \neg \mathit{Fragile}(x) \supset \mathit{Fragile}(x)\}$. Then the following are logical consequences of

$$\Sigma' \wedge \boldsymbol{O}(\Sigma_0 \wedge \Box\beta) \wedge \Box\Omega\Sigma_{\mathrm{def}} \ :$$

1. $\boldsymbol{B}\mathit{Fragile}(o)$;
2. $[\mathit{drop}(o)]\boldsymbol{B}\mathit{Broken}(o)$;
3. $[\mathit{senseF}(o)]\boldsymbol{K}\neg \mathit{Fragile}(o)$;
4. $[\mathit{senseF}(o)][\mathit{drop}(o)]\boldsymbol{K}\neg \mathit{Broken}(o)$.

(1) holds because of the default, since o's non-fragility is not yet known. Similarly, (2) holds because the default also applies after $\mathit{drop}(o)$. In particular, Theorem 4 applies as $[\mathit{drop}(o)]\boldsymbol{O}(\mathit{Broken}(o) \equiv \mathit{Fragile}(o) \wedge \Box\beta)$ follows as well. Finally, in (3) and (4) the agent has found out that o is not fragile, blocking the default since $\models \Box(\boldsymbol{K}\alpha \supset \boldsymbol{B}\alpha)$.

4 Related Work

While the situation calculus has received a lot of attention in the reasoning about action community, there are, of course, a number of alternative formalisms, including close relatives like the fluent calculus (Thielscher, 1999) and more distant cousins described in (Kowalski and Sergot, 1986; Gelfond and Lifschitz, 1993; Sandewall, 1994).

While $\mathcal{ES}_0$ is intended to capture a fragment of the situation calculus, it is also related to the work formalizing action and change in the framework of dynamic logic (Harel, 1984). Examples are (De Giacomo and Lenzerini, 1995) and later (Herzig *et al*, 2000), who also deal with belief. While these approaches remain propositional, there are also first-order treatments such as (Demolombe, 2003; Demolombe, Herzig, and Varzinczak, 2003; Blackburn, *et al* 2001), which, like $\mathcal{ES}_0$, are inspired by the desire to capture fragments of the situation calculus in modal logic. Demolombe (2003) even considers a form of only-knowing, which is related to the version of only-knowing in (Lakemeyer and Levesque, 2004), which in turn derives from the logic $\mathcal{OL}$ (Levesque and Lakemeyer, 2001).

The idea of progression is not new and lies at the heart of most planning systems, starting with STRIPS (Fikes and Nilsson, 1971), but also in implemented agent programming languages like 3APL (Hindriks *et al,* 1999). Lin and Reiter (1997) so far gave the most general account. Restricted forms of LR-progression, which are first-order definable, are discussed in (Lin and Reiter, 1997; Liu and Levesque, 2005; Claßen *et al,* 2007; Vassos and Levesque, 2007).

Default reasoning has been applied to actions mostly to solve the frame problem (Shanahan, 1993). Here, however, we use Reiter's monotonic solution to the frame problem (Reiter, 1991) and we are concerned with the static "Tweety-flies" variety of defaults. Kakas et al. (2008) recently made a proposal that deals with these in the presence of actions, but only in a propositional setting of a language related to $\mathcal{A}$ (Gelfond and Lifschitz, 1993).

5 Conclusion

The paper introduced a new semantics for the concept of only-knowing within a modal fragment of the situation calculus. In particular, we showed that, provided an agent starts with a basic action theory as its initial knowledge base, then all the agent knows after an action is again a basic action theory. The result is closely related to Lin and Reiter's notion of progression and generalizes it to allow for actions which return sensing results. We also showed how to handle static defaults in the sense that these are applied every time after an action has been performed. Because of the way only-knowing is modelled, defaults behave as in autoepistemic logic. In previous work we showed that by modifying the semantics of only-knowing in the static case, other forms of default reasoning like Reiter's default logic can be captured (Lakemeyer and Levesque, 2006). We believe that these results will carry over to our dynamic setting as well.

References

Blackburn, P., Kamps, J., Marx, M.: Situation calculus as hybrid logic: First steps. In: Brazdil, P.B., Jorge, A.M. (eds.) EPIA 2001. LNCS, vol. 2258, pp. 253–260. Springer, Heidelberg (2001)

Claßen, J., Eyerich, P., Lakemeyer, G., Nebel, B.: Towards an integration of Golog and planning. In: Veloso, M.M. (ed.) Proc. of IJCAI 2007, pp. 1846–1851 (2007)

Fikes, R., Nilsson, N.: STRIPS: A new approach to the application of theorem proving to problem solving. Artificial Intelligence 2, 189–208 (1971)

De Giacomo, G., Lenzerini, M.: PDL-based framework for reasoning about actions. In: Gori, M., Soda, G. (eds.) AI*IA 1995. LNCS, vol. 992, pp. 103–114. Springer, Heidelberg (1995)

Demolombe, R.: Belief change: from Situation Calculus to Modal Logic. In: IJCAI Workshop on Nonmonotonic Reasoning, Action, and Change (NRAC 2003), Acapulco, Mexico (2003)

Demolombe, R., Herzig, A., Varzinczak, I.J.: Regression in modal logic. J. of Applied Non-Classical Logics 13(2), 165–185 (2003)

Enderton, H.: A Mathematical Introduction to Logic. Academic Press, New York (1972)

Gelfond, M., Lifschitz, V.: Representing action and change by logic programs. Journal of Logic Programming 17, 301–321 (1993)

Harel, D.: Dynamic Logic. In: Gabbay, D., Guenther, F. (eds.) Handbook of Philosophical Logic, vol. 2, pp. 497–604. D. Reidel Publishing Company (1984)

Herzig, A., Lang, J., Longin, D., Polacsek, T.: A logic for planning under partial observability. In: Proc. AAAI-2000, AAAI Press, Menlo Park (2000)

Hindriks, K.V., De Boer, F.S., Van der Hoek, W., Meyer, J.-J.C.: Agent programming in 3APL. Autonomous Agents and Multi-Agent Systems 2(4), 357–401 (1999)

Hughes, G., Cresswell, M.: An Introduction to Modal Logic. Methuen and Co., London (1968)

Kakas, A., Michael, L., Miller, R.: Fred meets Tweety. In: Proc. ECAI-2008, pp. 747–748. IOS Press, Amsterdam (2008)

Kowalski, R., Sergot, M.: A logic based calculus of events. New Generation Computing 4, 67–95 (1986)

Lakemeyer, G., Levesque, H.J.: Situations, si! Situation Terms, no! In: Ninth Conf. on Principles of Knowledge Representation and Reasoning, AAAI Press, Menlo Park (2004)

Lakemeyer, G., Levesque, H.J.: A useful fragment of the situation calculus. In: Proc. of IJCAI-2005, pp. 490–496. AAAI Press, Menlo Park (2005)

Lakemeyer, G., Levesque, H.J.: Towards an axiom system for default logic. In: Proc. of AAAI-2006, AAAI Press, Menlo Park (2006)

Lakemeyer, G., Levesque, H.J.: A semantical account of progression in the presence of defaults. In: Proc. of IJCAI-2009, AAAI Press, Menlo Park (2009)

Levesque, H.J.: All I Know: A Study in Autoepistemic Logic. Artificial Intelligence 42, 263–309 (1990)

Levesque, H.J., Lakemeyer, G.: The Logic of Knowledge Bases. MIT Press, Cambridge (2001)

Levesque, H.J., Reiter, R., Lespérance, Y., Lin, F., Scherl, R.B.: Golog: A logic programming language for dynamic domains. Journal of Logic Programming 31, 59–84 (1997)

Lin, F., Reiter, R.: How to progress a database. Artificial Intelligence 92, 131–167 (1997)

Liu, Y., Levesque, H.J.: Tractable reasoning with incomplete first-order knowledge in dynamic systems with context-dependent actions. In: Proc. of IJCAI-2005 (2005)

McCarthy, J., Hayes, P.J.: Some philosophical problems from the standpoint of artificial intelligence. Machine Intelligence 4, 463–502 (1969)

Moore, R.C.: Semantical considerations on nonmonotonic logic. Artificial Intelligence 25, 75–94 (1985)

Mylopoulos, J.: TORUS - A Natural Language Understanding System For Data Management. In: IJCAI, pp. 414–421 (1975)

Reiter, R.: The frame problem in the situation calculus: A simple solution (sometimes) and a completeness result for goal regression. In: Lifschitz, V. (ed.) Artificial Intelligence and Mathematical Theory of Computation, pp. 359–380. Academic Press, London (1991)

Reiter, R.: Knowledge in Action: Logical Foundations for Describing and Implementing Dynamical Systems. MIT Press, Cambridge (2001)

Sandewall, E.: Features and Fluents. The Representation of Knowledge about Dynamical Systems. Oxford University Press, Oxford (1994)

Shanahan, M.: Solving the Frame Problem. MIT Press, Cambridge (1997)

Scherl, R.B., Levesque, H.J.: Knowledge, action, and the frame problem. Artificial Intelligence 144(1-2), 1–39 (2003)

Thielscher, M.: From situation calculus to fluent calculus: State update axioms as a solution to the inferential frame problem. Artificial Intelligence 111(1–2), 277–299 (1999)

Vassos, S., Levesque, H.J.: Progression of situation calculus action theories with incomplete information. In: Proc. IJCAI 2007 (2007)

Social Modeling and *i**

Eric S. Yu

Faculty of Information, University of Toronto
Toronto, Canada M5S 3G6

Abstract. Many different types of models are used in various scientific and engineering fields, reflecting the subject matter and the kinds of understanding that is sought in each field. Conceptual modeling techniques in software and information systems engineering have in the past focused mainly on describing and analyzing behaviours and structures that are implementable in software. As software systems become ever more complex and densely intertwined with the human social environment, we need models that reflect the social characteristics of complex systems. This chapter reviews the approach taken by the *i** framework, highlights its application in several areas, and outlines some open research issues.

1 Why Social Modeling

In many scientific and engineering disciplines, the principles, premises, and objectives of the field are embedded in and manifested through the models that are the daily conceptual tools of the profession. The models reflect the kinds of understanding that is sought by practitioners of the field. In software and information systems engineering, the dominant modeling constructs have revolved around static relationships (as in entity-relationships models and class diagrams) and dynamic and behavioural properties (as in process models and state-based formalisms). This focus is understandable as conceptual models are ultimately translated into data and operations for machine execution. For a system to be successful however, it must function within the context of its environment. As the need to model and characterize the machine's environment was increasingly recognized, these same modeling techniques and formalisms have been extended to cover the world in which the machine operates, and how the machine interacts with that world. The world was thus largely seen through the lens of the mechanistic operations of computers.

In a keynote speech in 1997, Professor John Mylopoulos identified four main classes of modeling ontologies that would be crucial "in the time of the revolution." Static and dynamic ontologies were well developed and widely adopted. Two new kinds of modeling – intentional and social – were needed to respond to the emerging needs of the information revolution [81].

Few would have predicted the way the revolution has unfolded. In 1997, the Netscape browser was still a novelty, and the world-wide web was hardly a household name. XML had not yet been introduced. Computer use, especially in the information systems area, was dominated by business applications within organizations, with trained users in a job setting. Today computer use is all but taken for granted. RFID, GPS, online banking and shopping are everywhere. New generations grow up unable to imagine life without Google, Wikipedia, Facebook, instant messaging, or texting. Back in the work world, organizations are adopting Enterprise 2.0, playing catch up with the Web 2.0 services that their employees and patrons have already taken for granted in their personal and social life. The revolution continues, with new technologies and services emerging all the time – e-book readers, location-based services, digital paper, and so on. Information technologies and systems are impacting people's lives in deeper ways than ever before. Every innovation has the potential to bring benefits, as well as threats to privacy, livelihoods, and even cherished cultural values. In principle, the possibilities for good are limitless. The concerns and risks are also very real. What methods and techniques can software and information systems professionals use to deal with these questions? Social modeling is more relevant than ever before.

The *i** modeling framework [122][123] was an attempt to introduce some aspects of social modeling and reasoning into information system engineering methods, especially at the requirements level. Unlike traditional systems analysis methods which strive to abstract away from the people aspects of systems, *i** recognizes the primacy of social actors. Actors are viewed as being intentional, i.e., they have goals, beliefs, abilities, and commitments. The analysis focuses on how well the goals of various actors are achieved given some configuration of relationships among human and system actors, and what reconfigurations of those relationships can help actors advance their strategic interests. The *i** framework has stimulated considerable interest in a socially-motivated approach to systems modeling and design, and has led to a number of extensions and adaptations.

This chapter aims to present an overview of the ideas behind the *i** framework, some of the main application areas, and discusses some possible future directions.

2 Premises and Features of *i** Modeling

From the earliest days, there have been concerns about the pervasive impacts that computing technology was having on society (e.g., [57]). The concerns included humanistic, ethical, as well as pragmatic ones – as many technically sound systems fell into disuse, or met with resistance from users [70]. Studies on the social impact of computing have raised awareness and sensitivity to the potentially negative as well as positive impacts of technology on people's lives. However, it has been difficult to make social understanding and analysis an integral part of the mainstream system development process.

The *i** modeling approach is an attempt to bring social understanding into the system engineering process by putting selected social concepts into the core of the daily activity of system analysts and designers, i.e., by adopting a social ontology for the main modeling constructs. Social analysis would not be an adjunct to technical analysis, but would be the basis for driving the entire system development.

To overcome the limitations of the mechanistic worldview, we shift our attention away from the usual focus on activities and information flows. Instead, we ask: what does each actor want? How do they achieve what they want? And who do they depend on to achieve what they want? In the following, we review each of the main premises of *i** [114][115], and discuss how they are manifested through the features of the modeling framework.

2.1 Actor Autonomy

We adopt as a premise that the social world is unknowable and uncontrollable to a large extent. From the viewpoint of conventional modeling, this may seem unintuitive and prohibiting. What is not knowable can hardly be modeled. Interestingly, this premise provides a way out of the usual mechanistic conception of the world.

In *i**, the central conceptual modeling construct is the actor. It is an abstraction which is used to refer to an active entity that is capable of independent action. Actors can be humans, hardware and software, or combinations thereof. Actors are taken to be inherently autonomous - their behaviours are not fully controllable, nor are they perfectly knowable.

This notion of autonomy is distinct from the one in artificial intelligence, where it refers to an advanced capability to be achieved by design and technological implementation. Autonomous agents in AI are artificial agents implemented in hardware and software which can act on their own without human intervention. In social modeling, we take actor autonomy to be a characteristic of the real-world social phenomena that system designers have to contend with.

2.2 Intentionality

Although the behaviour of actors are not fully knowable or controllable, they are nevertheless not completely random. To explain and characterize their behaviour, we attribute motivation and intent to actors. By modeling what actors intend to achieve, we obtain a higher level characterization without specifying their exact behaviour.

In *i** modeling, we focus on intentional properties and relationships rather than actual behaviour. By not describing behaviour directly, an intentional description offers a way to characterize actors that respects the autonomy premise. Conventional system modeling which offers only static and dynamic ontologies leads to an impoverished and mechanistic view of the world. Intentional modeling provides a richer expressiveness that is appropriate for a social conception of the world. By attributing intentionality, we can express *why* an actor undertakes certain actions, or prefers one alternative over another. An intentional ontology allows analysis of means-ends relationships and of the space of alternatives for each actor.

Various notions of actor are included in some non-intentional modeling frameworks and languages, e.g., in the form of stick figures in UML use case diagrams [85], and swim lanes in BPMN [5]. These actors are not intentional or autonomous, so are inadequate for social modeling. Recent work in goal modeling in requirements engineering (e.g., [108][96]) have developed intentional modeling ontologies, but have not emphasized the social dimension of intentionality. The name *i** stands for distributed intentionality, which puts intentionality within the context of social networks of autonomous actors.

2.3 Sociality

Social phenomena are arguably infinitely rich. The treatment that a modeling framework can provide is necessarily limited. Conceptual modeling frameworks aim to offer succinct representations of certain aspects of complex realities through a small number of modeling constructs, with the intent that they can support some kinds of analysis.

In *i**, we choose to focus on one aspect of being social – that the well-being of an actor depends on other actors. Actors depend on each other for goals to be achieved, tasks to be performed, and resources to be furnished. By depending on some other actor, the depender actor takes advantage of opportunities that are made available through dependee actors. For example, my life is made easier by mechanics who are able and willing to repair my car, even if I myself am not capable. At the same time, as I depend on someone else, I become vulnerable to not receiving what I expect from them. These dependencies are therefore strategic to the actors concerned because they can be beneficial or harmful to their well-being. Actors would choose what dependencies to have according to their judgement on the potential gains and losses from them.

In *i**, the *Strategic Dependency (SD) model* (Fig. 1) is a network of directed dependency relationships among actors. A dependency link indicates that one actor (the *depender*) depends on another (the *dependee*) for something (the *dependum*). Four types of dependencies are distinguished. If the dependum is stated as an assertion, it is called a *goal dependency*. The depender wants the dependee to make the assertion true, without specifying how it is to be achieved. If the dependum is stated as an activity, it is called a *task dependency*. The depender wants the dependee

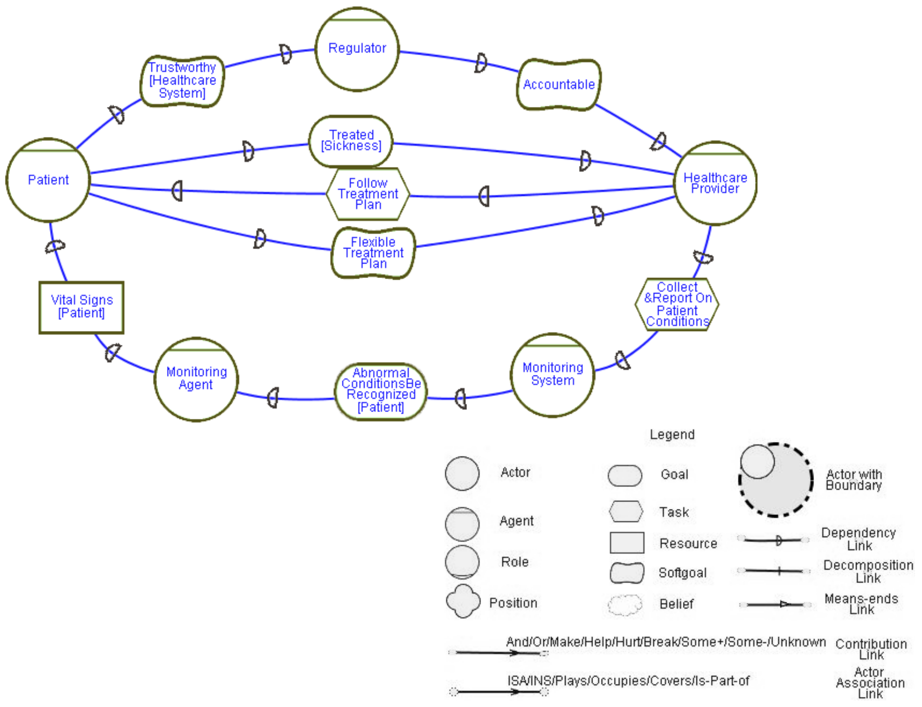


Fig. 1. A Strategic Dependency (SD) Model (from [114])

to perform the task as specified by the description of the activity. The dependency types therefore provide a way to convey the kinds of freedoms allowed in a relationship. In a goal dependency, the dependee is free to adopt any course of action, as long as the goal is achieved. The depender does not care how it is achieved. In a task dependency, the dependee's actions are confined, as stipulated by the depender. However, since the task description can seldom be complete and in full detail, the dependee still has freedom beyond what is specified.

A *resource dependency* is one in which the dependum is an entity, which can be an information or material object. The depender wants the dependee to furnish the entity so that it can be consumed as a resource, but is not concerned about the activity or any problem solving that may be needed to produce the entity.

A fourth type of dependency, called a *softgoal dependency*, is one in which the dependum is a quality, such as fast, cheap, reliable, secure, and so forth. A softgoal dependency is similar to a goal dependency except that the criteria for achievement of the quality goal is not sharply defined a priori, but may require elaboration and clarification. Consultation between depender and dependee may be required. The softgoal concept was first used to deal with non-functional requirements in software engineering [11]. It provides a useful mechanism for modeling many concepts in the social world which require contextual interpretation.

The SD model may be contrasted with process models employing dynamic ontologies. Unlike typical process models such as dataflow diagrams or activity diagrams which focus on information flow or control flow, the SD model is a higher level abstraction which depicts what actors want from each other, and the freedoms that each actor has. The SD model therefore acknowledges the autonomy of actors in a social world.

The SD model depicts external relationships among actors, while remaining silent regarding the internal makeup of the actors. For example, actors may possess knowledge that enables them to achieve goals and perform tasks, but this knowledge is not made explicit in the SD model.

Since an actor is autonomous, it may choose not to live up to the expectation from a depender. By analyzing the network of dependencies, one can infer that some dependencies are more likely to be viable than others. For example, if A depends on B for something, while B in turn has some substantive dependencies on A, directly or indirectly, then A's dependency is likely to be viable. Dependency failures may propagate along a chain of dependencies. When an actor (as dependee) is committed to delivering on a dependum, its efforts may prevent the failure from further propagating [122].

2.4 Rationality

The kinds of analyses that can be performed with the SD model are limited due to the strong assumption about actor autonomy. The SD model focuses on external relationships, while staying mute on internal makeup. In making sense of a social world, we often attribute motivations and intents to actors. We construct coherent explanations of their behaviour by relating their actions to attributed goals and motives.

In the *Strategic Rationale (SR) model* (Fig. 2), we attribute goals, tasks, resources, and softgoals to each actor, this time as internal intentional elements that the actor wants to achieve. A *means-ends link* is used to connect a task to a goal, indicating a specific way to achieve the goal. Typically there is more than one way to achieve a goal, so a goal in an SR model prompts the question – how else can this goal be achieved?

A task can be further specified through *task decomposition links* to indicate the subtasks, subgoals, resources, and softgoals that need to be performed or satisfied in order for the task to succeed.

Tasks have *contributions links* to softgoals indicating how they contribute to achieving those qualities (positively or negatively, with what strength). High level softgoals are refined into more specific softgoals through AND, OR combinations as well as partial contributions. Softgoals help distinguish among alternate tasks that can achieve the same goal, but differ in how they affect desired quality attributes. For example, the goal to Keep Well can be achieved through Patient-Centred Care or Provider-Centred Care, but the two options affect the patient's Lifestyle and Quality of Care differently. A qualitative reasoning procedure is used to propagate labels across the graph to evaluate goal achievement. When goals are not sufficiently met, actors look for further alternatives that produce more favourable outcomes.

In attributing rationality to actors in an *i** model, we are not asserting that these actors are intrinsically rational. Rather, rationality is externally attributed so that we as analysts can reason about their behaviour. In accordance with the autonomy premise, the model is inherently incomplete and may well be inaccurate.

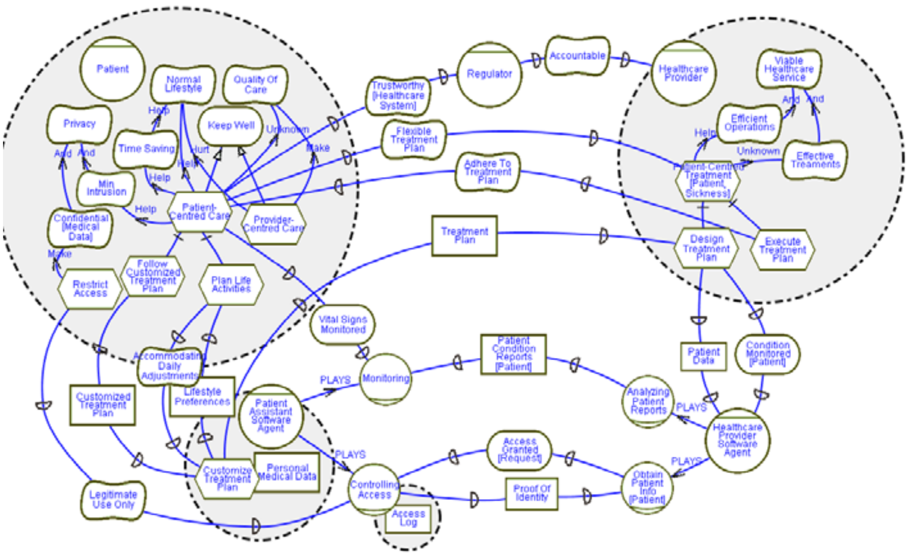


Fig. 2. A Strategic Rationale (SR) Model (from [114])

A belief is an assertion held to be true by an actor. It is useful for noting assumptions or justifications which when revoked, should trigger re-evaluation of affected decisions. Unlike a goal, the actor does not aim to make a belief come true.

2.5 Contingent Boundaries and Identities

Since we take the actor to be a modeling abstraction, its identity and scope are determined by the modeler. For example, it is up to the modeller to model persons in a work group individually as actors, or the entire group as an actor, or each person supported by software tools as an actor. Each of these would offer different opportunities for analysis. Organizations and communities acting coherently can also be treated as actors. Actors can have selfish as well as altruistic goals.

In *i**, the relationship that an actor has with other actors serves to demarcate the boundary of the actor at an intentional level. When an actor delegates a responsibility to another actor, the inter-actor boundaries shift accordingly.

Social modeling needs to deal with physically embodied actors such as humans as well as abstract "logical" actors such as roles. In *i**, the term *agent* is used to refer to actors with physical embodiment. An agent may play multiple *roles*. A set of roles typically played by one agent is called a *position*.

2.6 Strategic Reflectivity

In conventional systems analysis, the models typically provide an operational level description of the system, e.g., the information and control flows. The activities and reasoning used to improve the operation of the system, e.g., how to improve efficiency, reliability, cost, security, etc., are typically done outside of the models. An intentional ontology, such as the use of goal models (e.g., [11][108] would make the intentional dimension explicit, allowing trade-offs across multiple competing or synergistic goals.

In a social ontology, such consideration and tradeoffs would be modelled from the viewpoint of each actor. Each actor reflects upon its relationships with other actors, and makes judgements about the merits of various configurations with respect to its own strategic interests.

In *i**, each operational configuration is typically expressed through an SD model. The alternatives that are explored in an SR model refer to the alternative SD configurations that have different implications for the various strategic interests held by each actor.

3 Social Modeling for Requirements Engineering

In system development, requirements engineering (RE) serves as the crucial interface between the world of the application domain and the world of computing technology.

Much effort has been devoted to developing requirements models and languages that can lead to precise specifications of what the system should do. A major milestone was the recognition that requirements need to be defined in relation to the environment [9][48]. Modeling the world was therefore as important as modeling the machine [38]. The dominant ontologies for requirements modeling were static and dynamic ontologies, centring around entities and relationships (what exists), activities and processes (what occurs), and assertions and constraints (what holds true), e.g., [38][85].

When the social environment is complex, social modeling offers a new kind of analysis that could not be achieved within the mechanistic world view of static and dynamic ontologies. *i** modeling and variants have been used to obtain requirements in domains as diverse as air traffic control [71], agriculture [90], healthcare [56][3], environmental sensing [33], e-government [19], submarine systems [89], telecommunications [2], industrial enterprise [91], and business processes (see next section).

Each of these environments involves many social actors with disparate interests, a multiplicity of roles, and complex networks of relationships. Modeling the relationships at an intentional level offers a higher level of abstraction for analysis. Intentional modeling can also include stakeholders who have no direct information flow or activity interaction with the operational system, such as regulators, investors, or the general public, who may nevertheless have an influence over the system.

By identifying relationships among stakeholders using the SD model, weaknesses in existing relationships can be revealed. The SR model provides a systematic way for exploring the space of alternatives for achieving stakeholder goals, including different ways of using technology systems. This type of analysis is important at the early stages of system conception, but is not supported by conventional requirements modeling techniques. We refer to this as "early" requirements engineering.

Intentional social modeling can be complemented with other techniques. In the RESCUE methodology [72], *i** is used in conjunction with scenario techniques, and is synchronized at several stages to cross-check requirements obtained. Creativity workshops are also used to complement the model-based techniques. The joint use of the Goal-oriented Requirements Language (GRL, a variant of *i**) and a scenario mapping approach is also promoted in the User Requirements Notation (URN), a newly approved ITU-T standard [46][2][63].

The social modeling of *i** leverages the techniques developed in goal-oriented RE, such as systematic refinement and the eventual operationalization of goals, and goal graph evaluation [11][42]. Both the goal-oriented and social approach fall within the category of problem-oriented RE as discussed by Wieringa [112], in contrast to solution-oriented RE techniques which aim to specify the target system, e.g., IEEE 830 [45] and UML [85]. In the intentional ontology of goal-oriented RE, the problem is characterized by a set of goals, solutions are explored and evaluated as alternative ways for achieving the goals. In the social perspective taken in *i**, there is no single unified view of "the problem". Each stakeholder has strategic interests to pursue, and they have limited knowledge of and control over each other. They seek to advance their interests by considering different configurations of dependency relationships.

4 Social Modeling for Software Development

Software processes are social too. Software development is a complex social activity. Despite advances in tools, software work continues to be labour and knowledge intensive. Large numbers of people with specialized roles and skills collaborate in the development and maintenance of software products and services. Numerous approaches and methodologies have been proposed on how to organize and guide the software process, ranging from heavily disciplined, tightly controlled methods to lightweight "people-centred" agile ones [4]. Compared to other more mature engineering disciplines, software work is still highly variable and difficult to organize and manage. Schedule and budget overruns are commonplace, and high quality is hard to achieve.

Various modelling techniques have been used to support software processes (e.g., [15][84]), with special emphases on workflow support, tool automation, and method reuse. Few modeling methods, however, have focused on the social aspects of software work. Social considerations such as cooperation and conflict, motivation and rewards, responsibility and control, knowledge sharing and reuse can be critical to the success of any software project.

*i** modeling has been used to bring out the human social dimension of software processes [120]. [8] and [22] provide examples of applications of social modeling to the maintenance process in industrial settings. Knowledge management issues are analyzed using *i** in [39] and [105].

Software is social too. Having introduced social models in requirements engineering, a more radical proposal is to use social modeling in the design and analysis of software itself. As software is increasingly distributed and network-based, large software systems are taking on the characteristics of social systems - being composed of units interacting with each other with relative autonomy and on a peer-to-peer basis rather than the traditional hierarchical authority and centralized oversight. Indeed, the agent-oriented approach to software engineering is emerging as a promising new paradigm for constructing software [41].

In the past, modeling methods have typically followed paradigm shifts that originated from programming. Thus structured analysis followed structured programming and structured design [17]. Object-oriented analysis adopted concepts from object-oriented programming and object-oriented design [12]. In agent-oriented software, social characteristics are metaphorically attributed to complex software interactions. However, social modeling as envisioned in [81] and exemplified in *i** is not meant to be metaphorical. Rudimentary and reductionist as it may be, the social modeling is intended to reflect aspects of social reality, not only of artificial systems. Complex software systems are social not only because they have characteristics that resemble human social systems, but because they are actually driven by complex human social systems. A truly innovative way to reconceptualise software development is therefore to see software as in fact social. Social modeling can be used throughout the software lifecycle, not just at the requirements level.

Tropos. The Tropos project, initiated by Professor Mylopoulos, adopts the social ontology of *i** at the early requirements level. As the development process progresses, features of the social ontology are retained as much as possible during late requirements and architectural design. Detailed design and implementation are done on an agent software platform such as Jack or Jadex [7]. Software components are treated as having social characteristics. Formal Tropos combines formal techniques such as model checking with the social model analysis of *i** [25].

*i** modeling has also been used to guide COTS package selection [24], database design [49], data warehousing [88] and business intelligence [92]. An empirical evaluation of the *i** framework in a model-based software generation environment was presented in [23].

5 Social Modeling for Enterprise Engineering

Business processes. A common pitfall in enterprise IT is the adoption of technology without a clear and detailed understanding of how the technology will contribute to business goals. The business process reengineering (BPR) movement [40] has been instrumental in highlighting this pitfall. The conception of a business process served as a focal point for interaction between business analysts and IT developers. The use of models to describe and analyze business processes became a centerpiece in enterprise information systems engineering, so that the business can be understood and analyzed before considering technology solutions. Most business process modeling techniques, from flowcharts to the recent BPMN [5], were adapted from system analysis, and inherit a mechanistic world view, albeit simplified to engage business participants. Main features include information flows, activity steps, branching and merging, etc.

Business process models that show concrete flows and behaviour are easy for business stakeholders to validate, and provide good starting points for technology implementation. However the static and dynamic ontologies employed only describe what happens, but cannot be used to explain why, or to explore alternatives. Social models such as *i** can be used not only to relate business processes to business goals, but to the goals of various stakeholders who would be affected by any change (e.g., customers, employees, regulators, investors, etc.) [119][121]. Taking the interests of a full range of stakeholders into account during the redesign of a business process is likely to lead to process innovations and technology systems that are more broadly accepted and viable [56][3]. Social factors such as power and conflict, often the sources of failures, can be brought in for systematic analysis as part of the system development process. The Strategic Rationale model in *i** supports reasoning about alternate process designs and social configurations [121].

Compared to other socially motivated modeling techniques (e.g., [75]), *i** attempts a deeper social ontology [116], incorporating concepts such as strategic dependencies and actor autonomy [16]. While social analysis using narrative text can be much more nuanced and therefore cannot be replaced by modeling, a model-based approach can provide more direct and traceable linkages to system development, making such social analysis more likely to have impact on technical system design and implementation. Some methods that start from *i** models leading to business processes execution include [62] [59] [53] [111] [29] [60]. An *i**-based method for process reengineering and system specification is developed in [37].

Enterprise architecture. As information systems multiply in organizations, systematic frameworks and approaches have been proposed to manage systems not one system at a time, but across the entire enterprise and beyond, dealing with issues such as interoperability and integration, governance and policy compliance.

Modelling is considered central in enterprise architecture, especially at the higher levels of abstraction for sharing systems-related knowledge across the enterprise (e.g., Zachman [125], ToGAF [86]). Most of the modeling relies on existing modeling methods, with static and dynamic ontologies. Some frameworks do emphasize the need for the modeling of "motivation" (column 6 in Zachman [125]). The Business Motivation Model, a recent OMG standard [83], has goals and means-ends relationships, but does not deal with social relationships. The use of *i** as intentional social models for enterprise architecture is suggested in [124]. Policy compliance using *i** based concepts are proposed in [28] [95].

Business model innovations and strategic change. Many business and industry sectors have been going through fundamental changes triggered by the Internet and now mobile technologies. eBay, Amazon, and Dell has been leading examples. The newspaper and publishing industries are seeing more dramatic transformations.

Conceptual modeling techniques have been used to describe and analyze business models [50], typically by introducing business specific ontologies, including such concepts as asset, revenue and value flow, channels, etc. Some prominent business authors have promoted graphical depictions of business goals [52].

Social modeling can complement these models by supporting analysis of strategic dependencies and analyzing alternative configurations that contribute differently to strategic business goals. The complementary use of *i** and the e^3value business modeling notation is outlined in [35]. An analysis of disruptive strategic change appears in [100]. Business model analysis leading to service-oriented system design was described in [66]. Business strategies of a networked value constellation were modelled using e^3value and a simplified version of *i** in [34].

6 Social Modeling for Security, Privacy, and Trust

Computer security has long been an active area of research. Many security models have been proposed. However, few have adopted a social perspective.

Security and privacy are ultimately human concerns. Despite advances in security and privacy technologies, breaches and failures continue to occur. Perfect security and privacy are acknowledged to be unattainable in practice [101]. Determined attackers have been able to overcome or bypass the strongest defensive mechanisms. Often, users themselves neglect or defeat the defensive measures when they interfere with work routines, or are too hard to use.

Social models allow the human issues of security, privacy, and trust to be systematically analyzed and addressed within an engineering process. In *i**, security, privacy, and trust can be modelled initially as high-level softgoals of some actors. Efforts to achieve them can be modelled in terms of refinement to more specific goals, such as confidentiality, integrity, availability, unlinkability, and so forth, eventually operationalizing them through implementable mechanisms such as encryption, firewalls, intrusion detectors, and anonymizers. The goals are accomplished via a network of hardware and software as well as human roles (security officers, network administrators, peer users, etc.) The dependencies among actors in such networks can be analyzed for viability, such as the adequacy or lack of reciprocal dependencies.

A social approach would recognize that security and privacy concerns are not necessarily high on every actor's agenda. They can be superseded by competing goals such as cost, task urgency, or convenience. An actor-based social model can reveal the trade-offs faced by each actor, this prompting system designers to seek solution alternatives that respond to the actor's overall needs and desires, not just those pertaining to security and privacy.

*i** modeling has been used to analyze and guide system design for security, privacy, and trust [118][117][21][93][99]. Goals and strategies of attackers (including insiders) can also be modelled and analyzed, to be taken into account during requirements analysis and design [65][21].

The Secure Tropos [77] approach added security specific constructs and introduced social ontology to security patterns [78]. Another line of work (also called Secure Tropos) provided extensions by defining ownership, permission, and delegation [30]. *i** has also been used as a starting point for deriving access controls [14]. Social modeling based on *i** were also applied to risk modeling and analysis [74].

In the TCD framework [26], *i** is used to model trust in a social network, with extensions to support quantitative simulation on actor behaviour, and changes over time in trust, distrust, and confidence in the network. A trust management framework which extends *i** by distinguishing delegation of permission from delegation of execution is described in [31]. [106] presents a cognitive model of trust expressed in an adapted *i** notation.

Intentional modeling ontologies, particularly goal models, have been developed for security requirements engineering [11][109]. The goal structures in these frameworks represent a single consolidated viewpoint, rather than distributed among multiple autonomous actors as in social models. Social modeling extends goal-based techniques by treating actors (such as users and attackers) as being autonomous but interdependent. Instead of finding best solutions in a graph structure from a single viewpoint, each actor seeks reconfigurations of social relationships that advance its strategic interests.

7 Research Issues

Social modeling, particularly in the form of *i** and variations, has been explored to some degree in research communities, mostly in the requirements engineering area. The preceding sections have highlighted selected work that use *i** or draw upon its basic concepts. Many have extended or adapted the basic *i** framework [122]. Industry adoption of social modeling remains limited. Most industry projects reporting experiences using *i** or related social modeling had close collaborations with academic researchers. Much remains to be done to make social modeling as widespread as static and dynamic modeling.

The *i** framework represents only one possible perspective on and approach for social modeling. It is hoped that many more new frameworks will emerge to allow a wide selection of modeling and analysis techniques, perhaps reflecting quite different underlying premises than those presented in section 2. In the following, a sampling of research issues arising from the *i** experience will be discussed. Many of these may be applicable to social modeling in general.

7.1 Usage Contexts and Methodologies

Formality. Conceptual models are abstractions which filter an understanding of the world through the lens of a small number of predefined concepts. Formal definition of the concepts, for example through an axiomatization in a formal language and logic will facilitate automated inference and tool support. A high degree of formality, however, requires specialized training and thus restricts the user population. *i** variants have ranged over a broad spectrum of formal to informal approaches.

Most widely used conceptual modeling notations – from Data Flow Diagrams to UML, are semi-formal, and rely heavily on graphical visual notations. Formality is more difficult to attain in social modeling, as there is little agreement on any precise characterization of social reality, or even its possibility and desirability.

How much formality, of what kind, for use in what context – these are crucial research questions to pursue in order to create practical social modeling methodologies. A simplified, fairly informal notation may be necessary to encourage untrained stakeholder participation and interaction, while a more formal version may be needed for greater expressive power, better tool support, larger scale projects, and more automated analysis. A similar approach is taken in BPMN [5].

Domain terms. Aside from the predefined modeling constructs, linguistic terms chosen by modellers to represent domain concepts can also present difficulties. To reflect stakeholder perspective and promote active participation and ownership of the models, faithfulness to the language used by stakeholder is important. On the other hand, to facilitate analysis and to share knowledge across projects, the analyst may need to rephrase the domain terms. In any case, where most visual presentations of conceptual models require concise phrases to embody a concept, the choice of an appropriate phrase that will accurately convey the intended meaning can be quite demanding. The adoption of a project lexicon or ontology [6] is worth considering. Methodologically, one may want to have different sets of models using different domain vocabularies, e.g., one set for stakeholder participation, another for sharing and reuse. Coordination among sets of models will be another research issue. Lightweight natural language processing may also be helpful [102].

Patterns. Creating models from scratch can be quite labour intensive. A common solution is to build up collections of reusable models or generic patterns. The pattern approach for *i** has been explored in a number of works [89][73][78][58]. Patterns represent generalized knowledge, so they must be re-contextualized when applied to a specific situation. There are questions of validity of a pattern, and of applicability to a specific circumstance. There is risk that reliance on available patterns may distort analysis of the unique circumstances of a specific situation [104]. *i** has also been used to formalize the representation of problem contexts, forces, and alternate solutions in design patterns [80].

Visual Presentation and Interaction. Graphical models rely on effective human interpretation, interactive manipulation, and visual analysis. Their visual and cognitive properties are emerging research topics [76].

One ongoing challenge in *i** modeling is model scalability. *i** models are inherently networks, reflecting its conception of multilateral social relationships. Strategic rationale models may have dominating tree structures, but softgoals can receive contributions from all levels in the decomposition hierarchy, resulting in general graph structures. It is therefore difficult to take advantage of hierarchical abstraction mechanisms that provide much of the structural simplification in traditional structured analysis techniques (e.g., [17]). These challenges can potentially be overcome by inventive use of view mechanisms [113][61][13] or aspect orientation [1].

7.2 Conceptual Limitations and Extensions

The reduction of a complex world into a small number of modeling concepts is necessarily a compromise – one is faced with tough choices on what to include or exclude. The original *i** framework reflected principles described in Section 2. In practice, some users have found *i** too simple and limited in expressiveness, requiring extensions to make further conceptual distinctions, especially in specialized areas such as security [30][21]. Others found it too complex, electing to use subsets of the *i** constructs, e.g., [19][34][43]. By comparison, DFDs and ER models have 3 to 4 main conceptual constructs, whereas UML and BPMN have many more. In this section, we consider some areas for further exploration.

Reasoning. Although the process of constructing a model can in itself contribute significantly to understanding the issues in a domain [20], a deeper understanding can be gained by analyzing the reasoning implied by the intentional structure of the model. The SR model in *i** is an explicit representation of means-ends reasoning and contributions to quality goals, albeit inherently partial and incomplete. The SD model provides pathways for propagating intentionality across actors.

A number of approaches have been developed for reasoning over goal models. The NFR framework [11] offers an interactive procedure for propagating labels across the NFR goal graph. Based on the link types and labels, propagation steps can be automated if they do not require human judgment, though they can be overridden manually if desired. Fully automated procedures have been developed [32], some using assigned weights and numerical values [97].

A combined use of interactive and automated methods is likely desirable. Interactive method with a high degree of human judgement may be best suited to early RE due to its participatory, informal nature, when the model is very incomplete and in the process of being iteratively elaborated [42]. A highly interactive procedure will engage the modeller more fully and contributes to understanding at every step. When the models are more stable in later stages, automated evaluation of goal satisfaction can greatly improve efficiency of the process. The semantics of goal models is an active research topic. More empirical studies are also warranted.

Beliefs, Assumptions, Justifications. While intentional ontologies have emphasized goals and goal-based reasoning, beliefs have not been so well investigated. Some goal modeling frameworks have given prominence to assumptions, justifications, and context (e.g., [54]). Beliefs appear in the NFR framework in the form of claims, and are subject to the same evaluation propagation procedure as softgoals. Further exploration of the semantics and implications for practical reasoning and analysis are needed in the context of social modeling.

Viewpoints. As outlined in the premises, actor autonomy implies that each actor is reasoning from its own perspective. Therefore the rationales of each actor are modeled separately, within its own boundary scope in the *i** SR model. This is in contrast to goal-oriented RE frameworks (such as KAOS or the NFR framework) which employ ontologies which are intentional but not social.

In the current formulation of the *i** framework, this premise is only partially supported, as the model admits only one perspective on each actor's reasoning. This is an oversimplification, given the premise that each actor has limited access to other actor's internal rationales. To fully adhere to the premise, each actor would have its own model of every other actor's rationales, i.e., we would need as many SR models as there are actors, each from one actor's viewpoint. The SR models could be the result of modeling from interview data obtained from each actor separately. A more elaborate methodology would provide guidelines and support for merging models, and how to manage multiple viewpoints to benefit from inconsistencies and disagreements in the process [98].

A related topic is the modeling of cross-cutting concerns. Aspect-oriented techniques have been used to extend the expressiveness of *i** [79] and to simplify *i** models [1].

Process dynamics. One fundamental question in the design of a modeling language is the extent to which various basic ontologies should be incorporated and how tightly they should be integrated. Under Structured Analysis, DFDs and ERDs addressed dynamic and static ontologies quite separately. A tighter integration of the two basic ontologies was one of the objectives of object-oriented frameworks. In *i**, the social ontology is closely tied to an intentional ontology of goals and rationales, but dynamic and static ontologies are not explicitly incorporated.

Lack of temporal concepts is often felt to be an inhibiting factor in understanding *i** models. When *i** is used to model a business process, only the social and intentional relationships are portrayed. There is no indication of the temporal progression of the process, no beginning or end. Separate models are needed to express the static relationships and dynamics.

Concerns with incorporating features from other ontologies into a social modeling framework include increased complexity, commitment to a single version of the other ontologies, and not being able to do justice to the other ontologies with a limited set of features. One approach is to provide a loose coupling with an existing language or notation that offers rich features based on other ontologies. The User Requirements Notation (URN) brings together the social and intentional modeling of *i** (in the form of GRL) and the scenario-oriented dynamic ontology of Use Case Maps (UCM). Mappings and linkages between the two ontologies are provided through a unified metamodel [46][63]. However, small extensions to a social ontology for a specific purpose can be effective. For example, [27] and [110] represent two approaches on temporal extensions to *i**, allowing process simulation. The former adds a *precedes* operator, while the latter approach adopts a fuller set of procedural operators from the ConGolog language (sequence, non-deterministic pick, test, repeat, etc.).

Evolution and Change. Most applications of social modeling are concerned about change – how to improve the social configuration to the benefit of stakeholders. The representation of change in *i** is limited. Alternate social configurations (e.g., "as-is", "to-be", "could-be", etc.) are typically depicted in separate SD models. An SR model can show multiple alternatives and how they contribute differently to various stakeholders' strategic interests, though the representation is limited by visual complexity.

The change from As-Is to To-Be is an abrupt structural change, with no representation or reasoning about the steps that may be needed to bring about the change. Changes in the environment, especially gradual continuous change, are hard to represent. Complementary use of system dynamics and social intentional modeling is a topic to be explored. A method for using *i** in adaptive system modeling has been proposed [33]. Modeling strategic change is studied in [100].

Roles. In complex social settings, a role is distinguished from the person who plays the role. In an organization, position occupied by a person typically covers multiples roles. For example, a project manager runs a project, but may or may not be the performance evaluator for employees. The distinctions are useful for separating intentional dependencies on a role from those on the agent that plays the role, and for identifying role conflicts. An organization can be modeled as an aggregate position, i.e., a set of positions related via dependencies, regardless of which individual persons are occupying those positions. Roles are also useful for analyzing security, emphasizing a social analysis perspective [64][65][31].

The *i** framework offers notations for distinguishing roles, agents, and positions, and the association links between them (plays, occupies, covers), but the meanings of these concepts are not well defined and remain open for interpretation [120].

Inheritance. Inheritance along a specialization dimension is an important mechanism in conceptual models and is heavily used in static and dynamic ontologies, especially in object-oriented modeling. Due to the premise of actor autonomy, the usual inheritance concept does not apply straightforwardly to intentional relationships among actors [64]. Some research issues are identified in [68]. In general, abstraction mechanisms (such as classification/instantiation, generalization/specialization, part-whole, etc.) that are well studied in conceptual modeling [81] for static and dynamic ontologies are not yet well developed for social modeling.

7.3 Model Management and Tools

Conceptual models serve multiple purposes. They may be used to facilitate communication – between analysts and stakeholders, among analysts, developers and project managements, within a project or across projects within an enterprise. They may be used to describe and understand existing situations, to uncover problems and issues, or to explore hypothetical scenarios and potential solutions. Some models are used in impromptu settings, such as sketches on a whiteboard. Others are meant to be records in a repository for later retrieval or reuse.

Social models produced for different purposes may well need different kinds of tool support and management methods. For many small scale applications of *i**, general purpose drawing tools such as Visio have been found to be adequate and flexible, with the advantage of broad availability, and not requiring special installation and learning.

About a dozen software tools have been developed to support *i** modeling and specialized functionalities. Several are open source, some based on the Eclipse platform (e.g., OpenOME [87], TAOM4E [107], jUCMNav [51], jPRIM [47]). Some have built on the programmability of general purpose tools, e.g., [67][94]. *i**-related tools and approaches are compared at the *i** wiki [44] and in [36][103].

With many extensions and variations, the diversity of metamodels for *i**-related notations and tools has arisen as a challenge. Proposals have been made to reconcile differences [69] and to have a common interchange format [10]. Integration with other modeling frameworks using a metamodeling approach using Telos [82] have been investigated [91][55]. When a model progresses through a series of versions, version management issues arise. Merging different versions of a model has been investigated [98]. View mechanisms have been studied in [113][61][13]. A commercial requirements management system (Telelogic DOORS) has been used as a repository to manage change across multiple modeling frameworks in [97] and [28].

8 Conclusions

As computing and information systems interact more intricately with the social world, social modeling has arisen as a new area for conceptual modeling. Experiences with the *i** framework have revealed encouraging possibilities as well as many research challenges.

Conceptual modeling is of course only one way to bring understanding of complex social phenomena into the system design process. Techniques such as participatory design, ethnography, and others can equally enrich the process of system design. Conceptual modeling approaches have the potential of a more direct integration into established system engineering methods, supporting fine-grained analysis and traceability. As social modeling evolves, much can be gained by further exploring the synergies between conceptual modeling methods and the rich understanding of the human social experience from the social sciences and humanities.

Acknowledgements. My sincere thanks goes to everyone who has contributed to the development of *i**, directly or indirectly, by using, adapting, extending, and debating it, and uncovering its limitations and shortcomings. Most of all, I am deeply indebted to Professor John Mylopoulos, whose vision and open-mindedness made possible my forays into social modeling. Funding sources for the research include the Natural Sciences and Engineering Research Council (NSERC) of Canada and Bell University Laboratories. I am also grateful to Neil Maiden, Daniel Amyot, and Lin Liu, who kindly provided feedback on a draft of this paper on very short notice. Figures 1 and 2 are taken from [114], with kind permission of Springer Science+Business Media.

References

1. Alencar, F., Castro, J., Moreira, A., Araújo, J., Silva, C., Ramos, R., Mylopoulos, J.: Integration of Aspects with i* Models. In: Kolp, M., Henderson-Sellers, B., Mouratidis, H., Garcia, A., Ghose, A.K., Bresciani, P. (eds.) AOIS 2006. LNCS, vol. 4898, pp. 183–201. Springer, Heidelberg (2008)
2. Amyot, D.: Introduction to the User Requirements Notation: Learning by Example. Computer Networks 42(3), 285–301 (2003)
3. An, Y., Dalrymple, P.W., Rogers, M., Gerrity, P., Horkoff, J., Yu, E.: Collaborative Social Modeling for Designing a Patient Wellness Tracking System in a Nurse-Managed Health Care Center. In: 4th Int. Conf. on Design Science Research in Information Systems and Technology (DESRIST) (2009)

4. Beck, K., Boehm, B.: Agility Through Discipline. IEEE Computer 44–46 (June 2003)
5. BPMN: Business Process Modeling Notation specification (2009), `http://www.bpmn.org`
6. Breitman, K., Leite, J.C.S.P.: Ontology as a Requirements Engineering Product. In: IEEE Int. Conf. Requirements Eng. RE 2003, pp. 309–319 (2003)
7. Bresciani, P., Perini, A., Giorgini, P., Giunchiglia, F., Mylopoulos, J.: TROPOS: an agent-oriented software development methodology. J. Autonomous Agents and Multiagent Systems 8(3), 203–236 (2004)
8. Briand, L.C., Yong-Mi Kim, Y.M., Melo, W.L., Seaman, C.B., Basili, V.R.: Q-MOPP: qualitative evaluation of maintenance organizations, processes and products. Journal of Software Maintenance 10(4), 249–278 (1998)
9. Bubenko, J.A.: Information Modeling in the Context of System Development. IFIP Congress, 395–411 (1980)
10. Cares, C., Franch, X., Perini, A., Susi, A.: iStarML: An XML-based Model Interchange Format for i*. In: Castro, J.B., Franch, X., Perini, A., Yu, E. (eds.) Proc. 3rd Int. i* Workshop, Recife, Brazil, February 11-12, 2008, vol. 322, pp. 13–16. CEUR Workshop Proceedings, CEUR-WS.org (2008)
11. Chung, L., Nixon, B.A., Yu, E., Mylopoulos, J.: Non-Functional Requirements in Software Engineering. Kluwer Academic Publishers, Dordrecht (1999)
12. Coad, P., Yourdon, E.: Object-Oriented Analysis, 2nd edn. Prentice-Hall, Englewood Cliffs (1991)
13. Cocca, C.: Towards Improved Visual Support for i* Modeling. MISt thesis. Faculty of Information, University of Toronto (2007)
14. Crook, R., Ince, D., Nuseibeh, B.: On modelling access policies: Relating roles to the organisational context. In: IEEE Int. Requirements Eng. Conf. RE 2005, pp. 157–166 (2005)
15. Curtis, W., Kellner, M.I., Over, J.: Process Modeling. Commun. ACM 35(9), 75–90 (1992)
16. Cysneiros, L.M., Yu, E.: Addressing Agent Autonomy in Business Process Management - with Case Studies on the Patient Discharge Process. In: Proc. of Information Resources Management Association Conference, New Orleans, pp. 436–439 (2004)
17. Demarco, T.: Structured Analysis and System Specification. Prentice-Hall, Englewood Cliffs (1979)
18. DesCARTES Architect. Catholic University of Louvain, Belgium, `http://www.isys.ucl.ac.be/descartes/`
19. Donzelli, P.: A goal-driven and agent-based requirements engineering framework. Requirements Engineering 9(1), 16–39 (2004)
20. Easterbrook, S.M., Yu, E., Aranda, J., Fan, Y., Horkoff, J., Leica, M., Qadir, R.A.: Do Viewpoints Lead to Better Conceptual Models? An Exploratory Case Study. In: IEEE Int. Requirements Eng. Conf., pp. 199–208 (2005)
21. Elahi, G., Yu, E.: A Goal Oriented Approach for Modeling and Analyzing Security Trade-Offs. In: Parent, C., Schewe, K.-D., Storey, V.C., Thalheim, B. (eds.) ER 2007. LNCS, vol. 4801, pp. 375–390. Springer, Heidelberg (2007)
22. Elahi, G., Yu, E., Annosi, M.C.: Modeling Knowledge Transfer in a Software Maintenance Organization - An Experience Report and Critical Analysis. In: Stirna, J., Persson, A. (eds.) PoEM 2008. LNBIP, vol. 15, pp. 15–29. Springer, Heidelberg (2008)
23. Estrada, H., Martínez, A., Pastor, O., Mylopoulos, J.: An Empirical Evaluation of the i* Framework in a Model-based Software Generation Environment. In: Dubois, E., Pohl, K. (eds.) CAiSE 2006. LNCS, vol. 4001, pp. 513–527. Springer, Heidelberg (2006)
24. Franch, X.: On the Lightweight Use of Goal-Oriented Models for Software Package Selection. In: Pastor, Ó., Falcão e Cunha, J. (eds.) CAiSE 2005. LNCS, vol. 3520, pp. 551–566. Springer, Heidelberg (2005)

25. Fuxman, A., Liu, L., Mylopoulos, J., Pistore, M., Roveri, M., Traverso, P.: Specifying and analyzing early requirements in Tropos. Requirements Engineering Journal 9(2), 132–150 (2004)
26. Gans, G., Jarke, M., Kethers, S., Lakemeyer, G.: Continuous requirements management for organisation networks: a (dis)trust-based approach. Requirements Engineering Journal 8(1), 4–22 (2003)
27. Gans, G., Jarke, M., Lakemeyer, G., Schmitz, D.: Deliberation in a metadata-based modeling and simulation environment for inter-organizational networks. Inf. Syst. 30(7), 587–607 (2005)
28. Ghanavati, S., Amyot, D., Peyton, L.: Towards a Framework for Tracking Legal Compliance in Healthcare. In: Krogstie, J., Opdahl, A.L., Sindre, G. (eds.) CAiSE 2007 and WES 2007. LNCS, vol. 4495, pp. 218–232. Springer, Heidelberg (2007)
29. Ghose, A., Koliadis, G.: Actor Eco-systems: From High-Level Agent Models to Executable Processes via Semantic Annotations. IEEE COMPSAC (2), 177–184 (2007)
30. Giorgini, P., Massacci, F., Mylopoulos, J., Zannone, N.: Modeling Security Requirements Through Ownership, Permission and Delegation. In: IEEE Int. Requirements Eng. Conf. RE 2005, France, pp. 167–176 (2005)
31. Giorgini, P., Massacci, F., Mylopoulos, J., Zannone, N.: Requirements engineering for trust management: model, methodology, and reasoning. Int. J. of Information Security 5(4), 25, 274 (2006)
32. Giorgini, P., Nicchiarelli, E., Mylopoulos, J., Sebastiani, R.: Formal Reasoning Techniques for Goal Models. Journal of Data Semantics 1, 1–20 (2003)
33. Goldsby, H.J., Sawyer, P., Bencomo, N., Cheng, B.H.C., Hughes, D.: Goal-based Modeling of Dynamically Adaptive System Requirements. In: 15th IEEE Int. Conf. on Engineering of Computer Based Systems, pp. 36–45 (2008)
34. Gordijn, J., Petit, M., Wieringa, R.: Understanding business strategies of networked value constellations using goal- and value modeling. In: IEEE Int. Conf. on Requirements Eng. RE 2006, pp. 126–135 (2006)
35. Gordijn, J., Yu, E., Van Der Raadt, B.: E-service design using i* and e3value modeling. IEEE Software 23(3), 26–33 (2006)
36. Grau, G., Cares, C., Franch, X., Navarrete, F.J.: A Comparative Analysis of i* Agent-Oriented Modelling Techniques. In: Int. Conf. on Software Eng. and Knowledge Eng., San Francisco Bay, California, USA, pp. 657–663 (2006)
37. Grau, G., Franch, X., Maiden, N.A.M.: PRiM: An i*-based process reengineering method for information systems specification. Inf. & Softw. Tech. 50(1-2), 76–100 (2008)
38. Greenspan, S.J., Mylopoulos, J., Borgida, A.: Capturing More World Knowledge in the Requirements Specification. In: ACM/IEEE Int. Conf. Softw. Eng., pp. 225–235 (1982)
39. Guizzardi, R.S.S.: Agent-oriented Constructivist Knowledge Management. Ph.D. thesis, Enschede: University of Twente. The Netherlands (2006)
40. Hammer, M.: Reengineering work: Don’t Automate, Obliterate. Harvard Business Review, pp. 104–112 (July 1990)
41. Henderson-Sellers, B., Giorgini, P. (eds.): Agent-Oriented Methodologies. Idea Group Inc., Hershey (2005)
42. Horkoff, J.: Using i* Models for Evaluation. M.Sc. Thesis, Dept. of Computer Science, University of Toronto (2006)
43. Horkoff, J., Elahi, G., Abdulhadi, S., Yu, E.: Reflective Analysis of the Syntax and Semantics of the i* Framework. In: Song, I.-Y., Piattini, M., Chen, Y.-P.P., Hartmann, S., Grandi, F., Trujillo, J., Opdahl, A.L., Ferri, F., Grifoni, P., Caschera, M.C., Rolland, C., Woo, C., Salinesi, C., Zimányi, E., Claramunt, C., Frasincar, F., Houben, G.-J., Thiran, P. (eds.) ER Workshops 2008. LNCS, vol. 5232, pp. 249–260. Springer, Heidelberg (2008)
44. i* wiki, `http://istar.rwth-aachen.de`
45. IEEE: Guide to Software Requirements Specifications. IEEE Standard 830-1993. In: Software Engineering Standards. IEEE Computer Society Press, Los Alamitos (1993)

46. International Telecommunications Union (ITU-T) Recommendation Z.151: User Requirements Notation (URN) - Language Definition (2008)
47. J-PRiM. A Process Reengineerng i* Modeling Tool, http://www.ideaciona.com/PhD/JPRIM/
48. Jackson, M.: System Development. Prentice-Hall, Englewood Cliffs (1983)
49. Jiang, L., Topaloglou, T., Borgida, A., Mylopoulos, J.: Goal-Oriented Conceptual Database Design. In: IEEE Int. Conf. on Requirements Eng., pp. 195-204 (2007)
50. Johannesson, P.: The Role of Business Models in Enterprise Modelling. In: Krogstie, J., et al. (eds.) Conceptual Modelling in Info. Systems Eng., pp. 123–140. Springer, Heidelberg (2007)
51. jUCMNav. University of Ottawa, http://jucmnav.softwareengineering.ca/jucmnav/
52. Kaplan, R.S., Norton, D.P.: Having trouble with your strategy? Then map it. Harvard Business Review, 167–176 (September-October 2002)
53. Kazhamiakin, R., Pistore, M., Roveri, M.: A Framework for Integrating Business Processes and Business Requirements. In: IEEE Int. Enterprise Distributed Object Computing Conf., pp. 9–20 (2004)
54. Kelly, T.P., McDermid, J.A.: A Systematic Approach to Safety Case Maintenance. In: Felici, M., Kanoun, K., Pasquini, A. (eds.) SAFECOMP 1999. LNCS, vol. 1698, pp. 13–26. Springer, Heidelberg (1999)
55. Kethers, S.: Multi-perspective modelling and analysis of cooperation processes. Doctoral dissertation, RWTH Aachen University, Germany (2000)
56. Kethers, S., Gans, G., Schmitz, D., Sier, D.: Modelling trust relationships in a healthcare network: Experiences with the TCD framework. In: Bartmann, D., et al. (eds.) European Conf. on Information Systems (ECIS), Regensburg, Germany, pp. 1321–1328 (2005)
57. Kling, R. (ed.): Computerization and Controversy: Value Conflicts and Social Choices, 2nd edn. Morgan Kaufmann, San Francisco (1996)
58. Kolp, M., Giorgini, P., Mylopoulos, J.: Organizational Patterns for Early Requirements Analysis. In: Eder, J., Missikoff, M. (eds.) CAiSE 2003. LNCS, vol. 2681, pp. 617–632. Springer, Heidelberg (2003)
59. Koubarakis, M., Plexousakis, D.: A formal framework for business process modelling and design. Information Systems 27(5), 299–319 (2002)
60. Lapouchnian, A., Yu, Y., Mylopoulos, J.: Requirements-Driven Design and Configuration Management of Business Processes. In: Alonso, G., Dadam, P., Rosemann, M. (eds.) BPM 2007. LNCS, vol. 4714, pp. 246–261. Springer, Heidelberg (2007)
61. Leica, M.F.: Scalability concepts for i* modeling and analysis. M.Sc. thesis. Dept. of Computer Science, University of Toronto (2005)
62. Lespérance, Y., Kelley, T., Mylopoulos, J., Yu, E.: Modeling dynamic domains with ConGolog. In: Jarke, M., Oberweis, A. (eds.) CAiSE 1999. LNCS, vol. 1626, pp. 365–380. Springer, Heidelberg (1999)
63. Liu, L., Yu, E.: Designing Information Systems in Social Context: A Goal and Scenario Modelling Approach. Information Systems 29(2), 187–203 (2004)
64. Liu, L., Yu, E., Mylopoulos, J.: Analyzing security requirements as relationships among strategic actors. In: Proc. 2nd symposium on requirements engineering for information security (SREIS 2002), Raleigh, North Carolina (2002)
65. Liu, L., Yu, E., Mylopoulos, J.: Security and privacy requirements analysis within a social setting. In: IEEE Int. Conf. on Requirements Eng. RE 2003, pp. 151–161 (2003)
66. Lo, A., Yu, E.: From Business Models to Service-Oriented Design: A Reference Catalog Approach. In: Parent, C., Schewe, K.-D., Storey, V.C., Thalheim, B. (eds.) ER 2007. LNCS, vol. 4801, pp. 87–101. Springer, Heidelberg (2007)
67. Lockerbie, J.A., Maiden, N.A.M.: REDEPEND: Extending i* Modelling into Requirements Processes. In: IEEE Int. Conf. on Requirements Eng., pp. 361–362 (2006)

68. López, L., Franch, X., Marco, J.: Defining Inheritance in i* at the Level of SR Intentional Elements. In: Castro, J.B., Franch, X., Perini, A., Yu, E. (eds.) Proc. 3rd Int. i* Workshop, Recife, Brazil. CEUR Workshop Proceedings, vol. 322, pp. 71–74. CEUR-WS.org (2008)
69. Lucena, M., Santos, E., Silva, C., Alencar, F., Silva, M.J., Castro, J.: Towards a unified metamodel for i*. In: IEEE Int. Conf. On Research Challenges in Information Science, RCIS 2008, pp. 237–246 (2008)
70. Lyytinen, K.: Different Perspectives on Information Systems: Problems and Solutions. ACM Computing Surveys 19(1), 5–46 (1987)
71. Maiden, N.A.M., Jones, S., Manning, S., Greenwood, J., Renou, L.: Model-Driven Requirements Engineering: Synchronising Models in an Air Traffic Management Case Study. In: Persson, A., Stirna, J. (eds.) CAiSE 2004. LNCS, vol. 3084, pp. 368–383. Springer, Heidelberg (2004)
72. Maiden, N.A.M., Jones, S.: The RESCUE Requirements Engineering Process: An Integrated User-Centred Requirements Engineering Process for Eurocontrol, Version 4.1 (2004), http://hcid.soi.city.ac.uk/research/Rescue.html
73. Maiden, N., Manning, S., Jones, S., Greenwood, J.: Generating Requirements from Systems Models Using Patterns: A Case Study. Requirements Eng. Journal 10(4), 276–288 (2005)
74. Matulevicius, R., Mayer, N., Mouratidis, H., Dubois, E., Heymans, P., Genon, N.: Adapting Secure Tropos for Security Risk Management during Early Phases of the Information Systems Development. In: Bellahsène, Z., Léonard, M. (eds.) CAiSE 2008. LNCS, vol. 5074, pp. 541–555. Springer, Heidelberg (2008)
75. Medina-Mora, R., Winograd, T., Flores, R., Flores, F.: The action workflow approach to workflow management technology. In: ACM Conf. on Computer-Supported Cooperative Work, Toronto, Canada, pp. 281–288 (1992)
76. Moody, D.L.: Cognitive Load Effects on End User Understanding of Conceptual Models: An Experimental Analysis. In: Benczúr, A.A., Demetrovics, J., Gottlob, G. (eds.) ADBIS 2004. LNCS, vol. 3255, pp. 129–143. Springer, Heidelberg (2004)
77. Mouratidis, H., Giorgini, P., Manson, G.: When security meets software engineering: A case of modelling secure information systems. Information Systems 30(8), 609–629 (2007)
78. Mouratidis, H., Weiss, M., Giorgini, P.: Security patterns meet agent oriented software engineering: A complementary solution for developing security information systems. In: Delcambre, L.M.L., Kop, C., Mayr, H.C., Mylopoulos, J., Pastor, Ó. (eds.) ER 2005. LNCS, vol. 3716, pp. 225–240. Springer, Heidelberg (2005)
79. Mussbacher, G.: Aspect-Oriented User Requirements Notation: Aspects in Goal and Scenario Models. In: Giese, H. (ed.) MODELS 2008. LNCS, vol. 5002, pp. 305–316. Springer, Heidelberg (2008)
80. Mussbacher, G., Amyot, D., Weiss, M.: Formalizing Patterns with the User Requirements Notation. In: Taibi, T. (ed.) Design Pattern Formalization Techniques, pp. 304–325. IGI Publishing (2007)
81. Mylopoulos, J.: Information Modeling in the Time of the Revolution. Inf. Syst. 23(3-4), 127–155 (1998)
82. Mylopoulos, J., Borgida, A., Jarke, M., Koubarakis, M.: Telos: Representing Knowledge about Information Systems. ACM Trans. on Information Systems 8(4), 325–362 (1990)
83. OMG. Business Motivation Model (BMM) (2006), http://www.omg.org/spec/BMM/
84. Object Management Group (OMG): SPEM: Software Process Engineering Metamodel, Version 2.0 (2008)
85. Object Management Group (OMG), Unified Modeling Language, http://www.uml.org
86. Open Group. The Open Group Architecture Framework. version 9 (2009), http://www.opengroup.org
87. OpenOME. University of Toronto, http://www.cs.toronto.edu/km/openome/

88. Pardillo, J., Trujillo, J.: Integrated Model-Driven Development of Goal-Oriented Data Warehouses and Data Marts. In: Li, Q., Spaccapietra, S., Yu, E., Olivé, A. (eds.) ER 2008. LNCS, vol. 5231, pp. 426–439. Springer, Heidelberg (2008)
89. Pavan, P., Maiden, N.A.M., Zhu, X.: Towards a Systems Engineering Pattern Language: Applying i* to Model Requirements Architecture Patterns. In: ICSE STRAW 2003: 2nd Int. Ws. From Software Requirements to Architectures, Portland, Oregon, USA, pp. 134–141 (2003)
90. Perini, A., Susi, A.: Developing a decision support system for integrated production in agriculture. Environmental Modelling and Software 19(9), 821–829 (2004)
91. Petit, M.: Formal Requirements Engineering of Manufacturing Systems: A Multi-Formalism and Component-Based Approach, PhD dissertation. University of Namur, Belgium (1999)
92. Pourshahid, A., Chen, P., Amyot, D., Forster, A.J., Ghanavati, S., Peyton, L., Weiss, M.: Toward an integrated User Requirements Notation framework and tool for Business Process Management. In: 3rd Int. MCeTech Conf. on eTechnologies, Montréal, Canada, pp. 3–15. IEEE Computer Society, Los Alamitos (2008)
93. Pourshahid, A., Tran, T.: Modeling Trust in E-Commerce: An Approach Based on User Requirements. In: 9th ACM Int. Conf. on Electronic Commerce (ICEC 2007), pp. 413–421 (2007)
94. REDEPEND-REACT. An Architecture Analysis Tool, `http://www.ideaciona.com/PhD/REDEPEND-REACT/`
95. Rifaut, R., Dubois, E.: Using Goal-Oriented Requirements Engineering for Improving the Quality of ISO/IEC 15504 based Compliance Assessment Frameworks. In: IEEE Int. Conf. on Requirements Eng. RE 2008, pp. 33–42 (2008)
96. Rolland, C.: Capturing System Intentionality with Maps. In: Krogstie, J., Opdahl, A.L., Brinkkemper, S. (eds.) Conceptual Modelling in Information Systems Engineering, pp. 141–158. Springer, Heidelberg (2007)
97. Roy, J.-F., Kealey, J., Amyot, D.: Towards Integrated Tool Support for the User Requirements Notation. In: Gotzhein, R., Reed, R. (eds.) SAM 2006. LNCS, vol. 4320, pp. 198–215. Springer, Heidelberg (2006)
98. Sabetzadeh, M., Easterbrook, S.: View merging in the presence of incompleteness and inconsistency. Requirements Engineering 11(3), 174–193 (2006)
99. Samavi, R., Topaloglou, T.: Designing Privacy-Aware Personal Health Record Systems. In: Song, I.-Y., Piattini, M., Chen, Y.-P.P., Hartmann, S., Grandi, F., Trujillo, J., Opdahl, A.L., Ferri, F., Grifoni, P., Caschera, M.C., Rolland, C., Woo, C., Salinesi, C., Zimányi, E., Claramunt, C., Frasincar, F., Houben, G.-J., Thiran, P. (eds.) ER Workshops 2008. LNCS, vol. 5232, pp. 12–21. Springer, Heidelberg (2008)
100. Samavi, R., Yu, E., Topaloglou, T.: Strategic Reasoning about Business Models: A Conceptual Modeling Approach. Information Systems and e-Business Management 7(2), 171–198 (2009)
101. Sandhu, R.S.: Good-Enough Security: Toward a Pragmatic Business-Driven Discipline. IEEE Internet Computing 7(1), 66–68 (2003)
102. Sawyer, P., Rayson, P., Cosh, K.: Shallow knowledge as an aid to deep understanding in early phase requirements engineering. IEEE Trans. on Softw. Eng. 31(11), 969–981 (2005)
103. Schmitz, D., Lakemeyer, G., Jarke, M.: Comparing TCD/SNet with two other formal analysis approaches based on i*: Formal Tropos and Secure Tropos. In: Latour, T., Petit, M. (eds.) 8th Workshop on Agent-Oriented Information Systems (AOIS@CAiSE), pp. 29–40. Presses Universitaires de Namur (2006)
104. Strohmaier, M., Horkoff, J., Yu, E., Aranda, J., Easterbrook, S.M.: Can Patterns Improve i* Modeling? Two Exploratory Studies. In: Paech, B., Rolland, C. (eds.) REFSQ 2008. LNCS, vol. 5025, pp. 153–167. Springer, Heidelberg (2008)

105. Strohmaier, M., Yu, E.S., Horkoff, J., Aranda, J., Easterbrook, S.M.: Analyzing Knowledge Transfer Effectiveness: An Agent-Oriented Modeling Approach. In: 40th Hawaii Int. Conf. on Sys. Sci. HICSS 2007, p. 188. IEEE Computer Society, Los Alamitos (2007)
106. Sutcliffe, A.G.: Trust: From Cognition to Conceptual Models and Design. In: Dubois, E., Pohl, K. (eds.) CAiSE 2006. LNCS, vol. 4001, pp. 3–17. Springer, Heidelberg (2006)
107. TOAM4E. Tool for Agent Oriented Modeling. FBK-IRST, Italy, `http://sra.itc.it/tools/taom4e/`
108. van Lamsweerde, A.: Goal-Oriented Requirements Engineering: A Guided Tour. In: 5th IEEE Int. Symp. on Requirements Eng. RE 2001, Toronto, pp. 249–263 (2001)
109. van Lamsweerde, A.: Elaborating Security Requirements by Construction of Intentional Anti-Models. In: 26th Int. Conf. on Software Eng. ICSE 2004, pp. 148–157. ACM/ IEEE, Edinburgh (2004)
110. Wang, X., Lespérance, Y.: Agent-oriented requirements engineering using ConGolog and i*. In: Wagner, G., Karlapalem, K., Lespérance, Y., Yu, E. (eds.) Agent-Oriented Information Systems Workshop (AOIS 2001), Montreal, Canada, pp. 59–78. iCue Publishing, Berlin (2001)
111. Weiss, M., Amyot, D.: Business process modeling with URN. International Journal of E-Business Research 1(3), 63–90 (2005)
112. Wieringa, R.: Requirements Engineering: Problem Analysis and Solution Specification. In: Koch, N., Fraternali, P., Wirsing, M. (eds.) ICWE 2004. LNCS, vol. 3140, pp. 13–16. Springer, Heidelberg (2004)
113. You, J.Z.: Using meta-schema driven views for scaling i* models. M.Sc. thesis. Dept. of Computer Science, University of Toronto (2004)
114. Yu, E.: Agent-Oriented Modelling: Software Versus the World. In: Wooldridge, M.J., Weiß, G., Ciancarini, P. (eds.) AOSE 2001. LNCS, vol. 2222, pp. 206–225. Springer, Heidelberg (2002)
115. Yu, E.: Agent Orientation as a Modelling Paradigm. Wirtschaftsinformatik 43(2), 123–132 (2001)
116. Yu, E.S.K.: Models for Supporting the Redesign of Organizational Work. In: Conf. on Organizational Computing Systems (COOCS 1995), pp. 225–236. ACM Press, New York (1995)
117. Yu, E., Cysneiros, L.M.: Designing for Privacy in the Presence of Other Requirements. In: Falcone, R., Barber, S., Korba, L., Singh, M.P. (eds.) AAMAS 2002. LNCS, vol. 2631, pp. 209–223. Springer, Heidelberg (2003)
118. Yu, E., Liu, L.: Modelling Trust for System Design Using the *i** Strategic Actors Framework. In: Falcone, R., Singh, M., Tan, Y.-H. (eds.) Trust in Cyber-societies. LNCS, vol. 2246, pp. 175–194. Springer, Heidelberg (2001)
119. Yu, E.S.K., Mylopoulos, J.: From E-R to A-R: Modelling Strategic Actor Relationships for Business Process Reengineering. In: Loucopoulos, P. (ed.) ER 1994. LNCS, vol. 881, pp. 548–565. Springer, Heidelberg (1994)
120. Yu, E.S.K., Mylopoulos, J.: Understanding 'Why' in Software Process Modelling, Analysis, and Design. In: IEEE Int. Conf. Softw. Eng., pp. 159–168 (1994)
121. Yu, E.S.K., Mylopoulos, J.: Using Goals, Rules and Methods to Support Reasoning in Business Process Reengineering. Int. J. of Intelligent Systems in Accounting, Finance, and Management 5(1), 1–13 (1996)
122. Yu, E.S.K.: Modelling Strategic Relationships For Process Reengineering. Ph.D. dissertation. Dept. of Computer Science, University of Toronto (1995)
123. Yu, E.S.: Towards Modelling And Reasoning Support For Early-Phase Requirements Engineering. In: 3rd IEEE Int. Symp. on Requirements Eng., pp. 226–235 (1997)
124. Yu, E.S.K., Strohmaier, M., Deng, X.: Exploring Intentional Modeling and Analysis for Enterprise Architecture. In: Workshop on Trends in Enterprise Architecture Research (TEAR), 10th IEEE Int. Enterprise Distributed Object Computing Conference, October 2006, pp. 32.1– 32.8. IEEE Comp. Soc., Los Alamitos (2006)
125. Zachman, J.A.: A Framework for Information Systems Architecture. IBM Systems Journal 26(3) (1987)

Data Modeling in Dataspace Support Platforms

Anish Das Sarma[1], Xin (Luna) Dong[2], and Alon Y. Halevy[3]

[1] Stanford University
anish@cs.stanford.edu
[2] AT&T Labs-Research, New Jersey, USA
lunadong@research.att.com
[3] Google Inc.
halevy@google.com

Abstract. Data integration has been an important area of research for several years. However, such systems suffer from one of the main drawbacks of database systems: the need to invest significant modeling effort upfront. Dataspace Support Platforms (DSSP) envision a system that offers useful services on its data without any setup effort, and improve with time in a pay-as-you-go fashion. We argue that in order to support DSSPs, the system needs to model uncertainty at its core. We describe the concepts of probabilistic mediated schemas and probabilistic mappings as enabling concepts for DSSPs.

1 Introduction

Data integration and exchange systems offer a uniform interface to a multitude of data sources, and the ability to share data across multiple systems. These systems have recently enjoyed significant research and commercial success [10,11]. Current data integration systems are essentially a natural extension of traditional database systems in that queries are specified in a structured form and data is modeled in one of the traditional data models (relational, XML). In addition, the data integration system has exact knowledge of how the data in the sources map to the schema used by the data integration system. Consequently, to set up a data integration application there is a need for significant upfront effort in creating the mediated schema and the schema mappings.

Dataspace Support Platforms (DSSP) [12] envision data integration systems where the amount of upfront effort is much smaller. The system should be able to bootstrap itself and provide some useful services with no human intervention. Over time, through user feedback or as sources are added and the data management needs become clearer, the system evolves in a *pay-as-you-go* fashion.

To support DSSPs, we cannot rely on the same data modeling paradigms that form the basis for data integration systems. In particular, we cannot assume that the mediated schema is given to us in advance and that the schema mappings between the sources and the mediated schema will be *accurate*. We argue that a DSSP needs to support uncertainty at its very core: both in the mediated schema and in the schema mappings.

This article describes some of the formal foundations for data integration with uncertainty. We define probabilistic schema mappings and probabilistic mediated schemas, and show how to answer queries in their presence. With these foundations, it is possible to completely automatically bootstrap a pay-as-you-go integration system.

This article is largely based on previous papers [3,5]. The proofs of the theorems we state and the experimental results validating some of our claims can be found therein.

2 Uncertainty in Data Integration

We begin by describing some of the possible sources of uncertainty in a dataspace-like data integration system, and then explain how such a system should differ from a traditional data integration system.

2.1 Sources of Uncertainty

Uncertainty can arise in several ways in a dataspace system.

Uncertain mediated schema: The mediated schema is the set of schema terms in which queries are posed. They do not necessarily cover all the attributes appearing in any of the sources, but rather the aspects of the domain that the application builder wishes to expose to the users. Uncertainty in the mediated schema can arise for several reasons. First, as we describe in Section 4, if the mediated schema is automatically inferred from the data sources during bootstrapping, there will be some uncertainty about the results. Second, when domains are broad, there will be some uncertainty about how to model them. For example, if we model all the topics in Computer Science there will be some uncertainty about the degree of overlap between different topics.

Uncertain schema mappings: Schema mappings specify the semantic relationships between the terms in the sources and the terms used in the mediated schema. However, schema mappings can be inaccurate. In a dataspace, we expect that many of the initial schema mappings will be automatically derived, and hence will be inaccurate. In many applications it is impossible to create and maintain precise mappings between data sources. This can be because the users are not skilled enough to provide precise mappings, such as in personal information management [6], because people do not understand the domain well and thus do not even know what correct mappings are, such as in bioinformatics, or because the scale of the data prevents generating and maintaining precise mappings, such as in integrating data on the scale of the web [16].

Uncertain data: Data sources in a dataspace may not always be well structured, and therefore some of the data may be obtained by automatic methods. Furthermore, systems that include many sources (e.g., the Web) may contain unreliable or inconsistent data. Even in enterprise settings, it is common for informational data such as gender, race, and income level to be dirty or missing, even when the transactional data is precise.

Uncertain queries: We expect that much of the early interaction with a dataspace will be through keyword queries, since the users are not aware of a (non-existent) schema. The system needs to translate these queries into some structured form so they can be reformulated with respect to the data sources. At this step, the system may generate multiple candidate structured queries and have some uncertainty about which query captures the real intent of the user.

2.2 System Architecture

We now describe the architecture of a data integration module for DSSPs and contrast it to a traditional data integration system.

The first and most fundamental characteristic of this system is that it is based on a probabilistic data model. This means that we attach probabilities to tuples that we process in the system, schema mappings, the mediated schemas, and possible interpretations of keyword queries posed to the system. In contrast, a traditional data integration system includes a single mediated schema and assumes a single (and correct) schema mapping between the mediated schema and each source. The data in the sources is also assumed to be correct.

Unlike a traditional data integration systems that assume that the query is posed in a structured fashion (i.e., can be translated to some subset of SQL), here we assume that queries are posed as keywords. Hence, whereas traditional data integration systems begin by reformulating a query onto the schemas of the data sources, a DSSP needs to first reformulate a keyword query into a set of candidate structured queries. We refer to this step as *keyword reformulation*. Note that keyword reformulation is different from techniques for keyword search on structured data (e.g., [14,1]) in that (a) it does not assume access to all the data in the sources or that the sources support keyword search, and (b) it tries to distinguish different structural elements in the query in order to pose more precise queries to the sources (e.g., realizing that in the keyword query "Chicago weather", "weather" is an attribute label and "Chicago" is an instance name). That being said, keyword reformulation should benefit from techniques that support answering keyword search on structured data.

The query answering model in a DSSP is also different. Instead of necessarily finding *all* answers to a given query, our goal is typically to find the top-k answers, and rank these answers most effectively.

The architecture of the system is shown in Figure 1. The system contains a number of data sources and a mediated schema (we omit probabilistic mediated schemas from this figure). When the user poses a query Q, which can be either a structured query

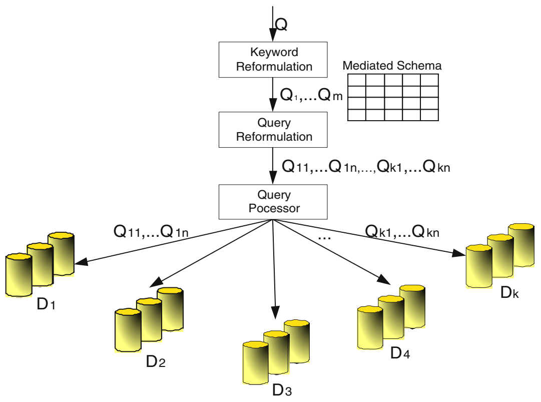


Fig. 1. Architecture of a data-integration system that handles uncertainty

on the mediated schema or a keyword query, the system returns a set of answer tuples, each with a probability. If Q is a keyword query, the system first performs keyword reformulation to translate it into a set of candidate structured queries on the mediated schema. Otherwise, the candidate query is Q itself.

We discuss probabilistic schema mappings in Section 3, and probabilistic mediated schemas in Section 4.

2.3 Source of Probabilities

A critical issue in any system that manages uncertainty is whether we have a reliable source of probabilities. Whereas obtaining reliable probabilities for such a system is one of the most interesting areas for research, there is quite a bit to build on. For keyword reformulation, it is possible to train and test reformulators on large numbers of queries such that each reformulation result is given a probability based on its performance statistics. In the case of schema matching, it is standard practice for schema matchers to also associate numbers with the candidates they propose [4,20]. The issue here is that the numbers are meant only as a ranking mechanism rather than true probabilities. However, as schema matching techniques start looking a larger number of schemas, one can ascribe probabilities (or approximations thereof) to their measures (see Section 3.4). Finally, information extraction techniques are also often based on statistical machine learning methods, thereby lending their predictions a probabilistic interpretation.

3 Uncertainty in Mappings

The key to resolving heterogeneity at the schema level is to specify schema mappings between data sources. These mappings describe the relationship between the contents of the different sources and are used to reformulate a query posed over one source (or a mediated schema) into queries over the sources that are deemed relevant. However, in many applications we are not able to provide all the exact schema mappings upfront. In this section we introduce probabilistic schema mappings (p-mappings) to capture uncertainty on mappings between schemas.

We start by presenting a running example for this section that also motivates p-mappings (Section 3.1). Then we present a formal definition of probabilistic schema mapping and its semantics (Section 3.2). Section 3.3 describes algorithms for query answering with respect to probabilistic mappings and discusses the complexity.

3.1 Motivating Probabilistic Mappings

Example 1. Consider a data source S, which describes a person by her email address, current address, and permanent address, and the mediated schema T, which describes a person by her name, email, mailing address, home address and office address:

```
S=(pname, email-addr, current-addr, permanent-addr)
T=(name, email, mailing-addr, home-addr, office-addr)
```

Possible Mapping	Prob
m_1 = {(pname, name), (email-addr, email), (current-addr, mailing-addr), (permanent-addr, home-addr)}	0.5
m_2 = {(pname, name), (email-addr, email), (permanent-addr, mailing-addr), (current-addr, home-addr)}	0.4
m_3 = {(pname, name), (email-addr, mailing-addr), (current-addr, home-addr)}	0.1

(a)

*p*name	*e*mail-addr	*c*urrent-addr	*p*ermanent-addr
Alice	alice@	Mountain View	Sunnyvale
Bob	bob@	Sunnyvale	Sunnyvale

(b)

Tuple (mailing-addr)	Prob
('Sunnyvale')	0.9
('Mountain View')	0.5
('alice@')	0.1
('bob@')	0.1

(c)

Fig. 2. The running example: (a) a probabilistic schema mapping between S and T; (b) a source instance D_S; (c) the answers of Q over D_S with respect to the probabilistic mapping

A semi-automatic schema-mapping tool may generate three possible mappings between S and T, assigning each a probability. Whereas the three mappings all map pname to name, they map other attributes in the source and the target differently. Figure 2(a) describes the three mappings using sets of attribute correspondences. For example, mapping m_1 maps pname to name, email-addr to email, current-addr to mailing-addr, and permanent-addr to home-addr. Because of the uncertainty about which mapping is correct, we consider all of these mappings in query answering.

Suppose the system receives a query Q composed on the mediated schema and asking for people's mailing addresses:

```
Q: SELECT mailing-addr FROM T
```

Using the possible mappings, we can reformulate Q into different queries:

```
Q1: SELECT current-addr FROM S
Q2: SELECT permanent-addr FROM S
Q3: SELECT email-addr FROM S
```

If the user requires all possible answers, the system generates a single aggregation query based on Q_1, Q_2 and Q_3 to compute the probability of each returned tuple, and sends the query to the data source. Suppose the data source contains a table D_S as shown in Figure 2(b), the system will retrieve four answer tuples, each with a probability, as shown in Figure 2(c).

If the user requires only the top-1 answer (i.e., the answer tuple with the highest probability), the system decides at runtime which reformulated queries to execute. For example, after executing Q_1 and Q_2 at the source, the system can already conclude that ('Sunnyvale') is the top-1 answer and can skip query Q_3. $\square$

3.2 Definition and Semantics

3.2.1 Schema Mappings

We begin by reviewing non-probabilistic schema mappings. The goal of a schema mapping is to specify the semantic relationships between a *source schema* and a *target schema*. We refer to the source schema as $\bar{S}$, and a relation in $\bar{S}$ as $S = \langle s_1, \dots, s_m \rangle$. Similarly, we refer to the target schema as $\bar{T}$, and a relation in $\bar{T}$ as $T = \langle t_1, \dots, t_n \rangle$.

We consider a limited form of schema mappings that are also referred to as *schema matching* in the literature. Specifically, a schema matching contains a set of *attribute correspondences*. An attribute correspondence is of the form $c_{ij} = (s_i, t_j)$, where s_i is a *source attribute* in the schema S and t_j is a *target attribute* in the schema T. Intuitively, c_{ij} specifies that there is a relationship between s_i and t_j. In practice, a correspondence also involves a function that transforms the value of s_i to the value of t_j. For example, the correspondence (c-degree, temperature) can be specified as temperature=c-degree $*1.8+32$, describing a transformation from Celsius to Fahrenheit. These functions are irrelevant to our discussion, and therefore we omit them. This class of mappings are quite common in practice and already expose many of the novel issues involved in probabilistic mappings. Broader classes of p-mappings are discussed in [5]. Formally, relation mappings and schema mappings are defined as follows.

Definition 1 (Schema Mapping). *Let $\bar{S}$ and $\bar{T}$ be relational schemas. A* relation mapping *M is a triple (S, T, m), where S is a relation in $\bar{S}$, T is a relation in $\bar{T}$, and m is a set of attribute correspondences between S and T.*

When each source and target attribute occurs in at most one correspondence in m, we call M a one-to-one relation mapping.

A schema mapping *$\overline{M}$ is a set of one-to-one relation mappings between relations in $\bar{S}$ and in $\bar{T}$, where every relation in either $\bar{S}$ or $\bar{T}$ appears at most once.* □

A pair of instances D_S and D_T *satisfies* a relation mapping m if for every source tuple $t_s \in D_S$, there exists a target tuple $t_t \in D_t$, such that for every attribute correspondence $(p, q) \in m$, the value of attribute p in t_s is the same as the value of attribute q in t_t.

Example 2. Consider the mappings in Example 1. The source database in Figure 2(b) (repeated in Figure 3(a)) and the target database in Figure 3(b) satisfy m_1. □

3.2.2 Probabilistic Schema Mappings

Intuitively, a probabilistic schema mapping describes a probability distribution over a set of *possible* schema mappings between a source schema and a target schema.

Definition 2 (Probabilistic Mapping). *Let $\bar{S}$ and $\bar{T}$ be relational schemas. A* probabilistic mapping (p-mapping)*, pM, is a triple $(S, T, \mathbf{m})$, where $S \in \bar{S}$, $T \in \bar{T}$, and $\mathbf{m}$ is a set $\{(m_1, Pr(m_1)), \dots, (m_l, Pr(m_l))\}$, such that*

- *for $i \in [1, l]$, m_i is a one-to-one mapping between S and T, and for every $i, j \in [1, l]$, $i \neq j \Rightarrow m_i \neq m_j$.*
- *$Pr(m_i) \in [0, 1]$ and $\sum_{i=1}^{l} Pr(m_i) = 1$.*

*p*name	*e*mail-addr	*c*urrent-addr	*p*ermanent-addr
Alice	alice@	Mountain View	Sunnyvale
Bob	bob@	Sunnyvale	Sunnyvale

(a)

*n*ame	*e*mail	*m*ailing-addr	*h*ome-addr	*o*ffice-addr
Alice	alice@	Mountain View	Sunnyvale	office
Bob	bob@	Sunnyvale	Sunnyvale	office

(b)

*n*ame	*e*mail	*m*ailing-addr	*h*ome-addr	*o*ffice-addr
Alice	alice@	Sunnyvale	Mountain View	office
Bob	email	bob@	Sunnyvale	office

(c)

Tuple (mailing-addr)	Prob
('Sunnyvale')	0.9
('Mountain View')	0.5
('alice@')	0.1
('bob@')	0.1

(d)

Tuple (mailing-addr)	Prob
('Sunnyvale')	0.94
('Mountain View')	0.5
('alice@')	0.1
('bob@')	0.1

(e)

Fig. 3. Example 3: (a) a source instance D_S; (b) a target instance that is by-table consistent with D_S and m_1; (c) a target instance that is by-tuple consistent with D_S and $< m_2, m_3 >$; (d) $Q^{table}(D_S)$; (e) $Q^{tuple}(D_S)$

A schema p-mapping*,* $\overline{pM}$*, is a set of p-mappings between relations in* $\bar{S}$ *and in* $\bar{T}$*, where every relation in either* $\bar{S}$ *or* $\bar{T}$ *appears in at most one p-mapping.* □

We refer to a non-probabilistic mapping as an *ordinary mapping*. A schema p-mapping may contain both p-mappings and ordinary mappings. Example 1 shows a p-mapping (see Figure 2(a)) that contains three possible mappings.

3.2.3 Semantics of Probabilistic Mappings

Intuitively, a probabilistic schema mapping models the uncertainty about which of the mappings in pM is the correct one. When a schema matching system produces a set of candidate matches, there are two ways to interpret the uncertainty: (1) a single mapping in pM is the correct one and it applies to all the data in S, or (2) several mappings are partially correct and each is suitable for a subset of tuples in S, though it is not known which mapping is the right one for a specific tuple. Figure 3(b) illustrates the first interpretation and applies mapping m_1. For the same example, the second interpretation is equally valid: some people may choose to use their current address as mailing address while others use their permanent address as mailing address; thus, for different tuples we may apply different mappings, so the correct mapping depends on the particular tuple.

We define query answering under both interpretations. The first interpretation is referred to as the *by-table* semantics and the second one is referred to as the *by-tuple* semantics of probabilistic mappings. Note that one cannot argue for one interpretation over the other; the needs of the application should dictate the appropriate semantics. Furthermore, the complexity results for query answering, which will show advantages for by-table semantics, should not be taken as an argument in the favor of by-table semantics.

We next define query answering with respect to p-mappings in detail and the definitions for schema p-mappings are the obvious extensions. Recall that given a query and an ordinary mapping, we can compute *certain answers* to the query with respect to the mapping. Query answering with respect to p-mappings is defined as a natural extension of certain answers, which we next review.

A mapping defines a relationship between instances of S and instances of T that are *consistent* with the mapping.

Definition 3 (Consistent Target Instance). *Let $M = (S, T, m)$ be a relation mapping and D_S be an instance of S.*

An instance D_T of T is said to be consistent with D_S and M, *if for each tuple $t_s \in D_S$, there exists a tuple $t_t \in D_T$, such that for every attribute correspondence $(a_s, a_t) \in m$, the value of a_s in t_s is the same as the value of a_t in t_t.* □

For a relation mapping M and a source instance D_S, there can be an infinite number of target instances that are consistent with D_S and M. We denote by $Tar_M(D_S)$ the set of all such target instances. The set of answers to a query Q is the intersection of the answers on all instances in $Tar_M(D_S)$.

Definition 4 (Certain Answer). *Let $M = (S, T, m)$ be a relation mapping. Let Q be a query over T and let D_S be an instance of S.*

A tuple t is said to be a certain answer of Q with respect to D_S and M, *if for every instance $D_T \in Tar_M(D_S)$, $t \in Q(D_T)$.* □

By-table semantics: We now generalize these notions to the probabilistic setting, beginning with the by-table semantics. Intuitively, a p-mapping pM describes a set of possible worlds, each with a possible mapping $m \in pM$. In by-table semantics, a source table can fall in one of the possible worlds; that is, the possible mapping associated with that possible world applies to the whole source table. Following this intuition, we define target instances that are *consistent with* the source instance.

Definition 5 (By-table Consistent Instance). *Let $pM = (S, T, \mathbf{m})$ be a p-mapping and D_S be an instance of S.*

An instance D_T of T is said to be by-table consistent with D_S and pM, *if there exists a mapping $m \in \mathbf{m}$ such that D_S and D_T satisfy m.* □

Given a source instance D_S and a possible mapping $m \in \mathbf{m}$, there can be an infinite number of target instances that are consistent with D_S and m. We denote by $Tar_m(D_S)$ the set of all such instances.

In the probabilistic context, we assign a probability to every answer. Intuitively, we consider the certain answers with respect to each possible mapping in isolation. The probability of an answer t is the sum of the probabilities of the mappings for which t is deemed to be a certain answer. We define by-table answers as follows:

Definition 6 (By-table Answer). *Let $pM = (S, T, \mathbf{m})$ be a p-mapping. Let Q be a query over T and let D_S be an instance of S.*

Let t be a tuple. Let $\bar{m}(t)$ be the subset of $\mathbf{m}$*, such that for each $m \in \bar{m}(t)$ and for each $D_T \in Tar_m(D_S)$, $t \in Q(D_T)$.*

Let $p = \sum_{m \in \bar{m}(t)} Pr(m)$. If $p > 0$, then we say (t, p) is a by-table answer of Q with respect to D_S and pM. □

By-tuple semantics: If we follow the possible-world notions, in by-tuple semantics, different tuples in a source table can fall in different possible worlds; that is, different possible mappings associated with those possible worlds can apply to the different source tuples.

Formally, the key difference in the definition of by-tuple semantics from that of by-table semantics is that a consistent target instance is defined by a mapping *sequence* that assigns a (possibly different) mapping in $\mathbf{m}$ to each source tuple in D_S. (Without losing generality, in order to compare between such sequences, we assign some order to the tuples in the instance).

Definition 7 (By-tuple Consistent Instance). *Let $pM = (S, T, \mathbf{m})$ be a p-mapping and let D_S be an instance of S with d tuples.*

An instance D_T of T is said to be by-tuple consistent with D_S and pM*, if there is a sequence $\langle m^1, \ldots, m^d \rangle$ such that d is the number of tuples in D_S and for every $1 \leq i \leq d$,*

- *$m^i \in \mathbf{m}$, and*
- *for the i^{th} tuple of D_S, t_i, there exists a target tuple $t_i' \in D_T$ such that for each attribute correspondence $(a_s, a_t) \in m^i$, the value of a_s in t_i is the same as the value of a_t in t_i'.* □

Given a mapping sequence $seq = \langle m^1, \ldots, m^d \rangle$, we denote by $Tar_{seq}(D_S)$ the set of all target instances that are consistent with D_S and seq. Note that if D_T is by-table consistent with D_S and m, then D_T is also by-tuple consistent with D_S and a mapping sequence in which each mapping is m.

We can think of every sequence of mappings $seq = \langle m^1, \ldots, m^d \rangle$ as a separate event whose probability is $Pr(seq) = \Pi_{i=1}^{d} Pr(m^i)$. If there are l mappings in pM, then there are l^d sequences of length d, and their probabilities add up to 1. We denote by $\mathbf{seq}_d(pM)$ the set of mapping sequences of length d generated from pM.

Definition 8 (By-tuple Answer). *Let $pM = (S, T, \mathbf{m})$ be a p-mapping. Let Q be a query over T and D_S be an instance of S with d tuples.*

Let t be a tuple. Let $\overline{seq}(t)$ be the subset of $\mathbf{seq}_d(pM)$, such that for each $seq \in \overline{seq}(t)$ and for each $D_T \in Tar_{seq}(D_S)$, $t \in Q(D_T)$.

Let $p = \sum_{seq \in \overline{seq}(t)} Pr(seq)$. If $p > 0$, we call (t, p) a by-tuple answer of Q with respect to D_S and pM. □

The set of by-table answers for Q with respect to D_S is denoted by $Q^{table}(D_S)$ and the set of by-tuple answers for Q with respect to D_S is denoted by $Q^{tuple}(D_S)$.

Example 3. Consider the p-mapping pM, the source instance D_S, and the query Q in the motivating example.

In by-table semantics, Figure 3(b) shows a target instance that is consistent with D_S (repeated in Figure 3(a)) and possible mapping m_1. Figure 3(d) shows the by-table answers of Q with respect to D_S and pM. As an example, for tuple t =('Sunnyvale'), we have $\bar{m}(t) = \{m_1, m_2\}$, so the possible tuple ('Sunnyvale', 0.9) is an answer.

In by-tuple semantics, Figure 3(c) shows a target instance that is by-tuple consistent with D_S and the mapping sequence $< m_2, m_3 >$. Figure 3(e) shows the by-tuple answers of Q with respect to D_S and pM. Note that the probability of tuple t=('Sunnyvale') in the by-table answers is different from that in the by-tuple answers. We describe how to compute the probabilities in detail in the next section. □

3.3 Query Answering

The following theorems are proved in [5] and consider by-table and by-tuple query answering in turn.

Theorem 1. *Let Q be an SPJ query and let $\overline{pM}$ be a schema p-mapping. Answering Q with respect to $\overline{pM}$ in by-table semantics is in PTIME in the size of the data and the mapping.* □

Theorem 2. *Let Q be an SPJ query and let $\overline{pM}$ be a schema p-mapping. The problem of finding the probability for a by-tuple answer to Q with respect to $\overline{pM}$ is #P-complete with respect to data complexity and is in PTIME with respect to mapping complexity.* □

Although by-tuple query-answering is hard, [5] shows several important restricted cases where it can still be done in polynomial time.

3.4 Creating P-Mappings

In [3] we address the problem of generating a p-mapping between a source schema and a target schema. We assume an input of a set of weighted correspondences between the source attributes and the target attributes. These weighted correspondences are created by a set of schema matching modules. There are two main challenges in creating p-mappings: (1) There may not be any p-mapping *consistent* with a given set of weight correspondences, and (2) when there is a consistent p-mapping, there may be multiple consistent p-mappings, and the question is which of them to choose. The algorithm we describe in [3] is based on normalizing weighted correspondences to ensure consistency, followed by choosing the p-mapping that maximizes the *entropy* of the probability assignment.

4 Uncertainty in Mediated Schema

The mediated schema is the set of schema terms (e.g., relations, attribute names) in which queries are posed. They do not necessarily cover all the attributes appearing in any of the sources, but rather the aspects of the domain that are important for the integration application. When domains are broad, and there are multiple perspectives on them (e.g., a domain in science that is constantly under evolution), then there will be uncertainty about which is the correct mediated schema and about the meaning of its terms. Also, when the mediated schema is created automatically by inspecting the sources in a pay-as-you-go system, there will be uncertainty about the mediated schema.

In this section we first motivate the need for probabilistic mediated schemas (p-med-schemas) with an example (Section 4.1). In Section 4.2 we formally define p-med-schemas and relate them with p-mappings in terms of expressive power and semantics of query answering.

4.1 P-Med-Schema Motivating Example

Let us begin with an example motivating p-med-schemas. Consider a setting in which we are trying to automatically infer a mediated schema from a set of data sources, where each of the sources is a single relational table. In this context, the mediated schema can be thought of as a "clustering" of source attributes, with similar attributes being grouped into the same cluster. The quality of query answers critically depends on the quality of this clustering. Because of the heterogeneity of the data sources being integrated, one is typically unsure of the semantics of the source attributes and in turn of the clustering.

Example 4. Consider two source schemas both describing people:

```
S1(name, hPhone, hAddr, oPhone, oAddr)
S2(name, phone, address)
```

In S2, the attribute phone can either be a home phone number or be an office phone number. Similarly, address can either be a home address or be an office address.

Suppose we cluster the attributes of S1 and S2. There are multiple ways to cluster the attributes and they correspond to different mediated schemas. Below we list a few (in the mediated schemas we abbreviate hPhone as hP, oPhone as oP, hAddr as hA, and oAddr as oA):

M1({name}, {phone, hP, oP}, {address, hA, oA})
M2({name}, {phone, hP}, {oP}, {address, oA}, {hA})
M3({name}, {phone, hP}, {oP}, {address, hA}, {oA})
M4({name}, {phone, oP}, {hP}, {address, oA}, {hA})
M5({name}, {phone}, {hP}, {oP}, {address}, {hA}, {oA})

None of the listed mediated schemas is perfect. Schema M_1 groups multiple attributes from S1. M_2 seems inconsistent because phone is grouped with hPhone while address is grouped with oAddress. Schemas M_3, M_4 and M_5 are partially correct but none of them captures the fact that phone and address can be either home phone and home address, or office phone and office address.

Even if we introduce probabilistic schema mappings, none of the listed mediated schemas will return ideal answers. For example, using M_1 prohibits returning correct answers for queries that contain both hPhone and oPhone because they are taken to be the same attribute. As another example, consider a query that contains phone and address. Using M_3 or M_4 as the mediated schema will unnecessarily favor home

Possible Mapping	Probability
{(name, name), (hP, hPP), (oP, oP), (hA, hAA), (oA, oA)}	0.64
{(name, name), (hP, hPP), (oP, oP), (oA, hAA), (hA, oA)}	0.16
{(name, name), (oP, hPP), (hP, oP), (hA, hAA), (oA, oA)}	0.16
{(name, name), (oP, hPP), (hP, oP), (oA, hAA), (hA, oA)}	0.04

(a)

Possible Mapping	Probability
{(name, name), (oP, oPP), (hP, hP), (oA, oAA), (hA, hA)}	0.64
{(name, name), (oP, oPP), (hP, hP), (hA, oAA), (oA, hA)}	0.16
{(name, name), (hP, oPP), (oP, hP), (oA, oAA), (hA, hA)}	0.16
{(name, name), (hP, oPP), (oP, hP), (hA, oAA), (oA, hA)}	0.04

(b)

Answer	Probability
('Alice', '123-4567', '123, A Ave.')	0.34
('Alice', '765-4321', '456, B Ave.')	0.34
('Alice', '765-4321', '123, A Ave.')	0.16
('Alice', '123-4567', '456, B Ave.')	0.16

(c)

Fig. 4. The motivating example: (a) p-mapping for S_1 and M_3, (b) p-mapping for S_1 and M_4, and (c) by-table query answers w.r.t. $\mathbf{M}$ and $\mathbf{pM}$. Here we denote {phone, hP} by hPP, {phone, oP} by oPP, {address, hA} by hAA, and {address, oA} by oAA.

address and phone over office address and phone or vice versa. A system with M_2 will incorrectly favor answers that return a person's home address together with office phone number. A system with M_5 will also return a person's home address together with office phone, and does not distinguish such answers from answers with correct correlations.

A probabilistic mediated schema will avoid this problem. Consider a probabilistic mediated schema $\mathbf{M}$ that includes M_3 and M_4, each with probability 0.5. For each of them and each source schema, we generate a probabilistic mapping (Section 3). For example, the set of probabilistic mappings $\mathbf{pM}$ for S_1 is shown in Figure 4(a) and (b).

Now consider an instance of S_1 with a tuple

```
('Alice', '123-4567', '123, A Ave.',
          '765-4321', '456, B Ave.')
```

and a query

```
SELECT name, phone, address
FROM People
```

The by-table answer generated by our system with respect to $\mathbf{M}$ and $\mathbf{pM}$ is shown in Figure 4(c). (As we describe in detail in the following sections, we allow users to compose queries using any attribute in the source.) Compared with using one of M_2 to M_5 as a mediated schema, our method generates better query results in that (1) it treats answers with home address and home phone and answers with office address and office phone equally, and (2) it favors answers with the correct correlation between address and phone number. □

4.2 Probabilistic Mediated Schema

Consider a set of source schemas $\{S_1, \dots, S_n\}$. We denote the attributes in schema $S_i, i \in [1, n]$, by $attr(S_i)$, and the set of all source attributes as $\mathcal{A}$. That is, $\mathcal{A} = attr(S_1) \cup \dots \cup attr(S_n)$. We denote a mediated schema for the set of sources $\{S_1, \dots, S_n\}$ by $M = \{A_1, \dots, A_m\}$, where each of the A_i's is called a *mediated attribute*. The mediated attributes are *sets* of attributes from the sources, i.e., $A_i \subseteq \mathcal{A}$; for each $i, j \in [1, m], i \neq j \Rightarrow A_i \cap A_j = \emptyset$.

Note that whereas in a traditional mediated schema an attribute has a name, we do not deal with naming of an attribute in our mediated schema and allow users to use any source attribute in their queries. (In practice, we can use the most frequent source attribute to represent a mediated attribute when exposing the mediated schema to users.) If a query contains an attribute $a \in A_i, i \in [1, m]$, then when answering the query we replace a everywhere with A_i.

A *probabilistic mediated schema* consists of a set of mediated schemas, each with a probability indicating the likelihood that the schema correctly describes the domain of the sources. We formally define probabilistic mediated schemas as follows.

Definition 9 (Probabilistic Mediated Schema). *Let* $\{S_1, \dots, S_n\}$ *be a set of schemas. A* probabilistic mediated schema (p-med-schema) for $\{S_1, \dots, S_n\}$ *is a set*

$$\mathbf{M} = \{(M_1, Pr(M_1)), \dots, (M_l, Pr(M_l))\}$$

where

- *for each* $i \in [1, l]$, M_i *is a mediated schema for* $S_1, \dots, S_n$, *and for each* $i, j \in [1, l], i \neq j$, M_i *and* M_j *correspond to different clusterings of the source attributes;*
- $Pr(M_i) \in (0, 1]$, *and* $\Sigma_{i=1}^{l} Pr(M_i) = 1$. □

Semantics of queries: Next we define the semantics of query answering with respect to a p-med-schema and a set of p-mappings for each mediated schema in the p-med-schema. Recall that answering queries with respect to p-mappings returns a set of answer tuples, each with a probability indicating the likelihood that the tuple occurs as an answer. We consider by-table semantics here. Given a query Q, we compute answers by first answering Q with respect to each possible mapping, and then for each answer tuple t summing up the probabilities of the mappings with respect to which t is generated.

We now extend this notion for query answering that takes p-med-schema into consideration. Intuitively, we compute query answers by first answering the query with respect to each possible mediated schema, and then for each answer tuple taking the sum of its probabilities weighted by the probabilities of the mediated schemas.

Definition 10 (Query Answer). *Let S be a source schema and $\mathbf{M} = \{(M_1, Pr(M_1)), \ldots, (M_l, Pr(M_l))\}$ be a p-med-schema. Let $\mathbf{pM} = \{pM(M_1), \ldots, pM(M_l)\}$ be a set of p-mappings where $pM(M_i)$ is the p-mapping between S and M_i. Let D be an instance of S and Q be a query.*

Let t be a tuple. Let $Pr(t|M_i), i \in [1, l]$, be the probability of t in the answer of Q with respect to M_i and $pM(M_i)$. Let $p = \Sigma_{i=1}^{l} Pr(t|M_i) * Pr(M_i)$. If $p > 0$, then we say (t, p) is a by-table answer with respect to $\mathbf{M}$ and $\mathbf{pM}$.

We denote all by-table answers by $Q_{\mathbf{M},\mathbf{pM}}(D)$. □

We say that query answers A_1 and A_2 are *equal* (denoted $A_1 = A_2$) if A_1 and A_2 contain exactly the same set of tuples with the same probability assignments.

Expressive power: A natural question to ask at this point is whether probabilistic mediated schemas provide any added expressive power compared to deterministic ones. Theorem 3 shows that if we consider *one-to-many* schema mappings, where one source attribute can be mapped to multiple mediated attributes, then any combination of a p-med-schema and p-mappings can be equivalently represented using a deterministic mediated schema with p-mappings, but may not be represented using a p-med-schema with deterministic schema mappings. Note that we can easily extend the definition of query answers to one-to-many mappings as one mediated attribute can correspond to no more than one source attribute.

Theorem 3. *The following two claims hold.*

1. *Given a source schema S, a p-med-schema $\mathbf{M}$, and a set of p-mappings $\mathbf{pM}$ between S and possible mediated schemas in $\mathbf{M}$, there exists a deterministic mediated schema T and a p-mapping pM between S and T, such that $\forall D, Q$: $Q_{\mathbf{M},\mathbf{pM}}(D) = Q_{T,pM}(D)$.*
2. *There exists a source schema S, a mediated schema T, a p-mapping pM between S and T, and an instance D of S, such that for any p-med-schema $\mathbf{M}$ and any set $\mathbf{m}$ of deterministic mappings between S and possible mediated schemas in $\mathbf{M}$, there exists a query Q such that $Q_{\mathbf{M},\mathbf{m}}(D) \neq Q_{T,pM}(D)$.* □

In contrast, Theorem 4 shows that if we restrict our attention to one-to-one mappings, then a probabilistic mediated schema *does* add expressive power.

Theorem 4. *There exists a source schema S, a p-med-schema $\mathbf{M}$, a set of one-to-one p-mappings $\mathbf{pM}$ between S and possible mediated schemas in $\mathbf{M}$, and an instance D of S, such that for any deterministic mediated schema T and any one-to-one p-mapping pM between S and T, there exists a query Q such that, $Q_{\mathbf{M},\mathbf{pM}}(D) \neq Q_{T,pM}(D)$.* □

Constructing one-to-many p-mappings in practice is much harder than constructing one-to-one p-mappings. When we are restricted to one-to-one p-mappings, p-med-schemas grant us more expressive power while keeping the process of mapping generation feasible.

4.3 Creating P-Med-Schemas

In [3], we study the problem of creating p-med-schemas. First the algorithm constructs a graph with source attributes as nodes, and weighted edges representing pairwise similarities. Then, it groups similarity edges into *certain* and *uncertain* based on their weights. Clusterings due to inclusion of all certain edges and combinations of inclusion/exclusion of uncertain edges constitute possible mediated schemas. Probabilities on each mediated schema are assigned based on *consistency* with respect to the data sources.

To enable users to see a single mediated schema, reference [3] also describes how to *consolidate* (without loss) the p-med-schema with one-to-one p-mappings into an equivalent single mediated schema with one-to-many p-mappings.

5 Related Work

Probabilistic Mappings: There have been various models proposed to capture uncertainty on mappings between attributes. [8] proposes keeping the top-K mappings between two schemas, each with a probability (between 0 and 1) of being true. [9] proposes assigning a probability for matching of every pair of source and target attributes. This notion corresponds to weighted correspondences described in Section 3.4.

Magnani and Montesi [17] have empirically shown that top-k schema mappings can be used to increase the recall of a data integration process and Gal [7] described how to generate top-k schema matchings by combining the matching results generated by various matchers. The probabilistic schema mappings we described above are different as they contain all possible schema mappings that conform to the schema matching results and assigns probabilities to these mappings to reflect the likelihood that each mapping is correct. Nottelmann and Straccia [19] proposed generating probabilistic schema matchings that capture the uncertainty on each matching step. The probabilistic schema mappings we create not only capture our uncertainty on results of the matching step, but also take into consideration various combinations of attribute correspondences and describe a *distribution* of possible schema mappings where the probabilities of all mappings sum up to 1.

Mediated Schemas: He and Chang [13] considered the problem of generating a mediated schema for a set of web sources. Their approach was to create a mediated schema that is statistically maximally *consistent* with the source schemas. To do so, they assume that the source schemas are created by a *generative model* applied to some mediated schema, which can be thought of as a probabilistic mediated schema. The probabilistic mediated schema we described in this chapter has several advantages in capturing heterogeneity and uncertainty in the domain. We can express a wider class of attribute clusterings, and in particular clusterings that capture attribute correlations. Moreover, we are able to combine attribute matching and co-occurrence properties for the creation of the probabilistic mediated schema, allowing for instance two attributes from one source to have a nonzero probability of being grouped together in the mediated schema. Also, the approach for p-med-schema creation described in this chapter is independent of a specific schema-matching technique, whereas the approach in [13] is tuned for constructing generative models and hence must rely on statistical properties of source schemas.

Magnani et al. [18] proposed generating a set of alternative mediated schemas based on probabilistic relationships between *relations* (such as an Instructor relation intersects with a Teacher relation but is disjoint with a Student relation) obtained by sampling the overlapping of data instances. Here we focus on matching attributes within relations. In addition, our approach allows exploring various types of evidence to improve matching and we assign probabilities to the mediated schemas we generate.

Chiticariu et. al. [2] studied the generation of multiple mediated schemas for an existing set of data sources. They consider multi-table data sources, not considered in this chapter, but explore interactive techniques that aid humans in arriving at the mediated schemas.

6 Conclusions

The investigation of data integration with uncertainty is only beginning. This chapter described some of the fundamental concepts on which such DSSP data integration systems will be built, but there is a lot more to do.

The main challenge is to build actual data integration systems that incorporate uncertainty and thereby uncover a new set of challenges, such as efficiency and understanding what are the common types of uncertainty that arise in data integration applications, so techniques can be tailored for these cases.

Our work focussed only on bootstrapping a pay-as-you-go integration system. The next challenge is to find methods to improve it over time (see [15] for a first work on doing so).

References

1. Agrawal, S., Chaudhuri, S., Das, G.: DBXplorer: A system for keyword-based search over relational databases. In: Proc. of ICDE, pp. 5–16 (2002)
2. Chiticariu, L., Kolaitis, P.G., Popa, L.: Interactive generation of integrated schemas. In: Proc. of ACM SIGMOD (2008)
3. Das Sarma, A., Dong, L., Halevy, A.: Bootstrapping pay-as-you-go data integration systems. In: Proc. of ACM SIGMOD (2008)
4. Doan, A., Halevy, A.Y.: Semantic integration research in the database community: A brief survey. AI Magazine 26(1), 83–94 (2005)
5. Dong, X., Halevy, A.Y., Yu, C.: Data integration with uncertainty. In: Proc. of VLDB (2007)
6. Dong, X., Halevy, A.Y.: A platform for personal information management and integration. In: Proc. of CIDR (2005)
7. Gal, A.: Why is schema matching tough and what can we do about it? SIGMOD Record 35(4), 2–5 (2007)
8. Gal, A., Modica, G., Jamil, H., Eyal, A.: Automatic ontology matching using application semantics. AI Magazine 26(1), 21–31 (2005)
9. Gal, A., Anaby-Tavor, A., Trombetta, A., Montesi, D.: A framework for modeling and evaluating automatic semantic reconciliation (2003)
10. Halevy, A.Y., Ashish, N., Bitton, D., Carey, M.J., Draper, D., Pollock, J., Rosenthal, A., Sikka, V.: Enterprise information integration: successes, challenges and controversies. In: SIGMOD (2005)
11. Halevy, A.Y., Rajaraman, A., Ordille, J.J.: Data integration: The teenage years. In: VLDB (2006)

12. Halevy, A.Y., Franklin, M.J., Maier, D.: Principles of dataspace systems. In: PODS (2006)
13. He, B., Chang, K.C.: Statistical schema matching across web query interfaces. In: Proc. of ACM SIGMOD (2003)
14. Hristidis, V., Papakonstantinou, Y.: DISCOVER: Keyword search in relational databases. In: Proc. of VLDB, pp. 670–681 (2002)
15. Jeffery, S., Franklin, M., Halevy, A.: Pay-as-you-go user feedback for dataspace systems. In: Proc. of ACM SIGMOD (2008)
16. Madhavan, J., Cohen, S., Dong, X., Halevy, A., Jeffery, S., Ko, D., Yu, C.: Web-scale data integration: You can afford to pay as you go. In: Proc. of CIDR (2007)
17. Magnani, M., Montesi, D.: Uncertainty in data integration: current approaches and open problems. In: VLDB workshop on Management of Uncertain Data, pp. 18–32 (2007)
18. Magnani, M., Rizopoulos, N., Brien, P., Montesi, D.: Schema integration based on uncertain semantic mappings. In: Delcambre, L.M.L., Kop, C., Mayr, H.C., Mylopoulos, J., Pastor, Ó. (eds.) ER 2005. LNCS, vol. 3716, pp. 31–46. Springer, Heidelberg (2005)
19. Nottelmann, H., Straccia, U.: Information retrieval and machine learning for probabilistic schema matching. Information Processing and Management 43(3), 552–576 (2007)
20. Rahm, E., Bernstein, P.A.: A survey of approaches to automatic schema matching. VLDB Journal 10(4), 334–350 (2001)

Conceptual Modeling: Past, Present and the Continuum of the Future

Nick Roussopoulos[1] and Dimitris Karagiannis[2]

[1] University of Maryland, Department of Computer Science, College Park, MD 20742, USA
nick@cs.umd.edu

[2] University of Vienna, Department of Knowledge and Business Engineering, Brünner Strasse 72, 1210 Vienna, Austria
dk@dke.univie.ac.at

Abstract. In this paper we give a brief history on conceptual modeling in computer science and we discuss state-of-the-art approaches. It is claimed that a number of problems remain to be solved. "Schema-first" is no longer a viable approach to meet the data needs in the dynamic world of the internet and web services. This is also true for the "schema-later" approach simply because data and data sources constantly change in depth and breadth and can neither wait for the schema to be completed nor for the data to be collected. As a solution we advocate for a new "schema-during" approach, in which the process of Conceptual Modeling is in a continuum with the operations in the database.

1 Introduction

Conceptual Modelling (CM) plays an important role in the development of software applications. Modelling acts as a starting point for understanding and, thus, forming the common basis for developers and users, freeing the stakeholders involved from implementation concerns. CM enables the integration of domain experts, who are involved in a business process, and their knowledge, into the software development.

Conceptual modelling is the activity of creating conceptual models, i.e., models that describe problems independently from the technology and strategy used to solve the problem [Ka08]. The term conceptual modelling was consolidated in Brodie, Mylopoulos and Schmidt [BMS84]. Mylopoulos [My92] defines CM as 'the activity of formally describing some aspects of the physical and social world around us for purposes of understanding and communication'. CMs play an important role in a variety of areas within computer science. Over the last decades it has found applications in a variety of fields, including information system design, knowledge representation for Artificial Intelligence, modelling of organizational environments, business processes, software development processes, software requirements, or just plain modelling some part of the world for purposes of human communication and understanding. Mylopoulos [My98] classifies CM according their domains and their application scenarios.

CM is an essential phase in the design of database systems [BLM92]. It dates back to the mid 70's, and its value has been well recognized, being considered by now as perhaps the most important component of database design. In databases, the original conceptual modeling methods were almost exclusively targeted towards the development of a global conceptual schema that would be the basis for understanding by all users and applications. It would provide the foundation for maintaining database consistency and integrity. The schema would also capture database access rules and data constraints, but nothing about the physical data layout and distribution. Data heterogeneity and database distribution were supposed to be completely hidden from the user, and a single layer, the conceptual schema, would provide "transparency" -- a term picked to mean "independence" from the underlying tedious details. The conceptual schema was built for the user by the design team, and the users had neither the need nor the ability to modify the schema it. They had to accept whatever predefined schema view or views were given to them.

The above "***schema-first***" approach became unrealistic with the vast explosion of data on the internet and web services. In this new environment, no modeler can design a global schema to satisfy the thousands or millions of users and/or user categories with predefined profiles. Furthermore, CM is a much slower process than web evolution. The other two approaches, "***schema-later***" **-** where the data is semi-structured, or "***schema-never***" **-** where all data is unstructured and irregular, are also unsatisfactory. Schema-later is worse than schema-first because it has to deal not only with the process of defining the schema but also with data transformation. Furthermore, it provides limited structural organization, limited access and optimization capability. Schema-never provides no efficient structured search, and is limited to what search engines like Google provide.

In this short paper, we discuss a completely new approach: ***"schema-during"*** (SD). The ideas are still at an early stage. However, we thought that providing some original albeit developing ideas would be most appropriate for the "very original" John Mylopoulos we honor in this symposium. Section 2 contains a short historical overview of CM, emphasizing its database connections. In section 3, we discuss the current state of conceptual modeling. Section 4 of this paper discusses SD and the environment it is intended for. It introduces the concept of "***sibling data***", a grouping mechanism for interrelated data linked together as a unit. Section 5 has the conclusions.

2 Conceptual Modeling - Past

Because a conceptual level view of a domain is not supposed to be distracted by implementation concerns – how the model will be represented inside the computer, CM is intimately related to the notion of *abstraction*, which also has a long and honorable history in Computer Science. In this section we highlight some of the chief achievements of both these areas, with no attempt at being complete, though we will try to point to the places where key contributions of our honoree, John Mylopoulos, took place.

In addition, a model of some application domain consists not just of static objects, but also dynamic events and activities that occur in it. We will therefore consider some CM techniques that also address these aspects.

2.1 Early 1970ies – The Roots of Conceptual Modelling

The following are some of the most significant precursors to CM, in fields outside of databases.

One of the first and most influential approaches to abstraction applied to *software development* can be found in [Pa72]. The major goal of the Parnas' approach is to provide sufficiently precise and complete specifications so that other pieces of software can be written to interact with it without additional information. This is achieved by providing concepts for data abstraction, by hiding implementation details from the user.

Another important path blazer in the field of modelling was the *programming language* Simula [DH72]. Simula introduced new concepts like objects, classes, methods and especially subclasses, which support the notion of generalization abstraction. Simula is considered the first object-oriented programming language, and became a cornerstone of most object-oriented techniques.

In *Artificial Intelligence*, the work of Quillian [Qu68] on semantic networks, a graphical knowledge representation language with nodes representing concepts and edges relationships laid, the cornerstone for much of the next decade's work, especially the exploration of the use of inheritance as an inference mechanism, which would become a hallmark of later CM techniques.

The Structured Analysis and Design Technique (SADT™), introduced by Ross [Ro77] in the mid seventies, was one of the most significant early steps in the area of *requirements specifications*. Among its distinguishing features is the equal emphasis on modelling data (connected by edges representing transformations) as well as activities (connected by edges representing flow of information.

2.2 Mid 1970ies – Databases: CM and Semantic Data Models

Most approaches to database desing initially relied on modelling the data structures used to store the model in actual file systems. Two noteworthy approaches are the hierarchical [TL76] and the network [Ts76] models, both focusing on the physical level – what nowdays might be called graph models [AG08].

Codd's revolutionary separation of logical data organization from physical organization laid the ground for CM and for capturing more of the semantics of an application [Co79]. Knowledge representation techniques as well as a form of data abstraction were introduced by Abrial [Ab74], who proposed defining the semantics of classes by access procedures. Roussopoulos & Mylopoulos [RM75] used a semantic network for generating a relational schema for the target system and in defining a set of semantic operators for maintaining the database consistent. Because of its simplicity, Chen's Entity-Relationship (ER) Model[1] [Ch76] became popular and the de facto standard in data modelling and database design. [HK87] provides a survey on the semantic data models.

Semantic data models allow for designing models at a higher level and enable the database practitioners to naturally and directly incorporate in the schema a larger portion of the semantics of the data [HM78]. Concepts for abstraction and generalisation in database design have been introduced in database research by Smith

[1] After the initial publication of Chen's E-R model, a large number of extensions to this model have been proposed in the eighties and nineties. Prominent examples are the Enhanced ER model (EER), the E^2R (read 'E-squared-R') and the Higher-Order Entity Relationship Model (HERM). The enhancements in the ER model introduce concepts like composite attribute, weak entity type, subclass and superclass relationships [HSA04].

and Smith [SS77]. To this point, database design concepts, as for example Codd's normal form [Co70], focused on abstraction, while the concept of generalisation had been largely ignored. Smith and Smith combined 'generalization' and 'aggregation' into one structuring discipline. However, aggregation was still not easily modelled using E-R; it became the main thrust in Object Oriented databases. The advantage of aggregation is that it provides an easier understanding of complex models and a more systematic approach to database design. It mainly supports the development of highly structured models without loss in intellectual manageability.

The first high level data definition languages for defining conceptual schemas – like the Conceptual Schema Language (CSL) - were discussed in the late seventies. Descriptive elements as well as procedural elements are provided within CSL. Hence, static aspects and dynamic behaviour of data could be described by providing standard types, object types and association types [BMF79].

A prominent example of a database design language, covering the aforementioned concepts, is Taxis [My78]. Taxis provides relational database management facilities, means of specifying semantic integrity constraints incorporated into transactions, and an exception-handling mechanism. Taxis applies the concepts of class, property and ISA (generalisation) relationship to all aspects of program design.

2.3 The 1980ies – An Efflorescense of CMs

In 1980, an influential workshop on Data Abstraction, Databases and Conceptual Modelling took place in Pingree Park. The purpose of the workshop was to overcome the boundaries between the sub-disciplines of computing as AI, Database Management and Programming Languages, ending up with a series of further joint initiatives to shape research and development of common interests. CM was introduced as a term reflecting this broader perspective. The specialisation has increased over the years and more and more sub-disciplines within CM emerged. Nowadays, it is nearly impossible to define, which of these specialisations developed over the years are required and which are accidental [Ka08]. However, the differences between those sub-disciplines seem to arise from issues concerning notation and basic vocabulary. Only in a minority of cases do the concepts introduced, the ways to utilise the models, or the ways the models are constructed justify this development.

In the field of Requirements Engineering, Greenspan et al [GM94] adopted the approach of Taxis (based on classes, properties and inheritance) and attempted to formalise the SADT notation, which took seriously Michael Jackson's prescription that requirements must connect the existing world with the proposed whole, by producing a model of the whole. RML embodies a notation for requirements modelling which combines object-orientation and organisation, with an assertional sublanuguage used to specify constraints and deductive rules, as well as time.

Another early example is the Adaptable Database Design methodology [RY84]. SDBD takes not only the environment of the data processing system into account, but also focuses on the environment of the entire enterprise. Proceeding on the assumption that without a complete understanding of how the enterprise operates, it is not possible to develop an effective design. Thus it is recommended to start with an environment analysis phase followed by a system analysis phase, capturing and analysing the operational behaviour of the organisation. [2]

2.4 The 1990ies – Object Oriented Development, a New Programming Paradigm

By the end of the eighties, a variety of object-oriented analysis techniques has been developed. Important representatives of these techniques are the 'Booch Method' and Rumbaugh's 'Object Modelling Technique', both offering a more coherent modelling framework than the combined use of data flow and entity-relationship diagrams (as for example proposed in SADT). The Booch Method focused mainly on object-oriented design (OOD), whereas the object-modelling technique (OMT) focused on object-oriented analysis (OOA). In 1994 Booch and Rumbaugh decided to combine and unify their object-oriented modelling methods by developing the Unified Modelling Language (UML) [OMG07] - a language for modelling object systems. Through the standardisation efforts undertaken by the Object Management Group (www.omg.org) UML has been rapidly adopted as the de facto standard for modelling a very wide range of applications and domains [BRJ05]. It is claimed that one important advantage of UML is that it could be used both for modelling software, and for modelling the problem domain that is supported by a system, i.e. conceptual modelling [EW05].

Another prominent example developed at that time is Telos – a knowledge representation language designed specifically for information systems development applications. The innovation of Telos is the treatment of attributes, promoted to a first class citizenship status, and the introduction of special representational and inferential mechanisms for handling temporal knowledge. These concepts provide means for dealing with the evolutionary nature of knowledge about software [My90], [My92].

By the end of the nineties it was widely agreed that information systems need to better match their operational organisational environment. Hence, requirement specification needs to cover not only software specifications but also business models and other kinds of information in describing the context in which the intended system will function. The above UML emphasises concepts for modelling and analysis during the later requirements phases, which usually focus on completeness, consistency, and automated verification of functional requirements [Al00]. With Tropos, a development method – based on the i* organisational modelling framework [Yu95] – supporting the early phases of the requirement capture is provided. Tropos is founded on the idea of using the agent paradigm and related mentalistic notions during all phases of the development software process [Su05].

3 Conceptual Modeling - Present

The rapid evolution of the internet and the World Wide Web has and will continue to bring about transformation. An increased need for integrating heterogeneous information sources has arisen, since organisations start to break away from usual application silo patterns and begin to develop a process-oriented view on the

[2] A reference implementation of the SDBD method is available on www.openmodels.at. The entire SDBD method is implemented on the meta modeling platform ADOxx 1.0.

organisations' business. More than that, software programs can be accessed and executed via the web based on the idea of 'Web Services' enabling distributed computation over the internet. Organisations are shifting towards a Service Oriented Architectures (SOA) but the real advantage of SOA will not develop as long as web services are not configurable at the business level [Ju08]. Hence, the gap between analysis and design phase has to be bridged.

To support this claim, existing conceptual modelling frameworks need to be reviewed and extended. As first attempts the integration of industry-wide business process modelling standards (as the 'Business Process Modelling Notation', BPMN) and the aforementioned web service standards can be seen. Business processes modelled in BPMN can be exported to BPEL for execution, allowing for the straight-through integration of business processes and web services. However, existing approaches for automating service recognition, service configuration and combination, service comparison and automated negotiation need to be enhanced and standardised.

Existing research has primarily focused on the technologies that support SOA deployments and the technical delivery of service based platforms [RLD08]. To standardize the use of web services, the web service technology builds on top of web standards and extends them with additional languages and protocols: The World Wide Web Consortium (W3C) proposed the Web Service Description Language (WSDL)[3] standard. The exchange of messages between web services is standardised via SOAP[4] and the publication and advertisement of web services is supported through service registries like UDDI[5]. In this context some work has been done to support the modelling of web services in extended UML and the automatic generation of the respective web service description in the WSDL standard [VdM05].

According to [Fe02], existing web technologies provide limited means in automating service recognition, service configuration and combination (i.e., providing complex workflows and business logics based on web services), service comparison and automated negotiation. To bridge this gap concepts and frameworks like the Web Service Modelling Framework (WSMF) [Fe02], that discusses an approach for facilitating usage and invocation as well as composition of web services, are promising approaches.

Other problems like the lack of concepts for the provision of service reputation mechanisms to provide ratings of the quality of services [MS02], and to solve the problem of ensuring trustworthiness between the stakeholders, are discussed in [MS02]. However a final solution on this topic is not offered.

A more general approach for interoperability on web-scale is the idea of the Semantic Web, where the contents of web resources, whether services or data, are to be described using formal domain ontologies that allow them to be compared and

[3] WSDL defines services as collections of network endpoints or ports. In WSDL the abstract definition of endpoints and messages is separated from their concrete network deployment or data format bindings (http://www.wsdl.org).

[4] SOAP is a message layout specification that defines a uniform way of passing XML-encoded data (http://www.soap.org).

[5] UDDI provides a mechanism for clients to find web services. Using a UDDI interface, businesses can dynamically lookup as well as discover services provided by external business partners (http://www.uddi.org).

matched. The current leading contenders for W3C approved languages to fulfil this role are RDFS[6] and OWL[7]. The latter is a Description Logic -- a descendant of semantic networks, which has been formalized and for which optimized sound and complete reasoners are now available.

4 Conceptual Modeling – The Continuum of the Future

In the world we live today, data is not hand-crafted but vastly generated by computers, data streams, data sensors, searches on the Web [Fr06] and other Service Oriented Architecture (SOA) applications. The data producers themselves constantly change; new ones are introduced or previous ones evolve in both content format and data density, i.e. more and higher resolution data, richer content, etc. What is needed is a new approach which we will call "***schema-during***" (SD), in which the conceptual modeling is in a "*continuum*" with the operational use of the database. In other words, the schema design is done continuously and concurrently with data loading, accessing, and other database management activities. The segregation of schema creation and database operations no longer exists. Both schema and data continuously evolve, and the end-user needs to be trained to manage both. The user also needs to be equipped with appropriate tools in order to manage schema evolution - an area that is still not well understood.

4.1 Conceptual Modeling for the Schema-During

We envision Conceptual Modeling to be a continuous process, performed concurrently with data loading and management. Since the web environment is constantly changing, modeling, populating, and managing data have to seamlessly integrated and be part of the database management. This schema-during (SD) activity brings a significant departure from the current practices. The end user will always have to adapt the conceptual schema to match the changes of the sources, and to track his interactions with the Web and SOA.

Data entry also occurs at any time, and is continuous. In many cases, raw data is found by accessing a wide variety of data sources, scraping the web, or by using a search engine or even accessing Wikis. Such raw data sources may be changing due to the discovery of new data sources or because the data is better processed/resolved, or simply because the sources changed their interface. Extract-Translate-Load (ETL) tools are necessary to help the end user to do data entry. Such tools exist for loading data warehouses and databases, but these were developed for programmers. New ETL tools need to be usable by the end user. The configurations within these tools will be for a short time before they have to be modified to adapt to environment changes. Hence these tools need to be flexible, easy to adapt, and user friendly in order to accommodate the continuous evolution of the data sources and the vast heterogeneity in them.

Conceptual modeling in an SD approach entails adaptation and personalization of the schema by the end user, which is in contrast to a steadfast global multi-user

[6] http://www.w3.org/RDF/
[7] http://www.w3.org/2004/OWL/

integrated schema developed by a competent database designer or the DBA team. If all users in SD are allowed to modify the schema, and evolve the data to comply with the new schema, a new set of problems will arise. Cooperative schema evolution in SD will open a new area of research, which would help the saturated database community. New research problems will arise when the database allows concurrent access and modification to the database schema. Transactional semantics need to be introduced for the intensional (schema) changes - the analog of transaction correctness (i.e. seriliazability). When concurrent transformations of the schema and data occur, the notion of "transformational seriliazability" is needed. In this paper we do not attempt to touch the issues of concurrent transformational semantics. First we need to understand and develop the techniques and transformation tools for such evolution, as for a single user. In the SD environment, the user needs to define his/her needs in terms of modeling by specifying the metadata that describes the data he wants to track. New entities and relationships are identified while existing ones need to be modified and extended.

We suggest the relational model and its tabular format to capture the user's models over a wide and heterogeneous collection of data sources. The reasons are very similar to those in the Polaris systems: simplicity, familiarity, etc. [St02]. Furthermore, the user needs to populate the table by mapping values appearing on the screen to the table row and column. The values will be extracted from a wide variety of underlying presentation layers such as HTML, Adobe pdf, Word, spreadsheet, graphical object, streams, etc. The underlying raw documents and their context need be preserved for visualization, further conceptual modeling and/or populating with data. In many cases, the tables may have to be expanded with new columns or reduced by deleting columns. In such cases, the interface has to support such an expansion/reduction, and allow the user to fill in the new columns with data. The transformations do not necessarily need to be complete -- some of the table columns may remain empty simply because the data was not available prior to the expansion. In an evolving world, we cannot insist on retrofitting or even inventing data that did not exist, just to smooth tables.

A Mouse Click Language

A very important tool for end-user conceptual modeling is to have a click-language for defining schema, data and populating the database. The ease of use of such a language is vital. In a Web service oriented environment, the user interacts with the browser and receives one or more screens full of data. The user can use the mouse and the keyboard to define metadata and data, or can highlight schema names on the screen, can right-click on the highlighted portions to bring a drop down menu to choose some action. These actions will include:

a) Metadata definition using the highlighted words or phrases. Those would then form the schema of a relation (the attribute names or the relation name) unordered. A template overlayed on the screen would allow him to reorder, modify, or simply rearrange these names.
b) Data entry into a previously defined table. Selection of the appropriate table can be given in a screen overlay with the options. Rearrangement of the values to the right attribute can be either done manually or could be inferred depending how "semantically strong" the values are or by remembering previous values of each column.

c) Repeated data entry is achieved by recording the screen actions (cursor positioning, click order, menu selections, etc) the user performs for a data entry. Naming, storing, and recalling such prerecorded repeated data entries is necessary even if it cannot always achieve full automation, and requires some manual adjustments.
d) Overlayed annotations to the highlighted parts of the screen. These are entered using the keyboard and are captured and glued to the screen position. Annotations may not be just text but any other multi-media data such as voice, images, video, URLs, and executable code. The code could possibly invoke other activities not necessarily related to database modeling.
e) Retrieval of data using a Query-By-Example (QBE) [Zl77] like interface. The values of the examples can also be picked from the highlighted portions of the screen or they can also be modified. A somewhat useful search query would be where a value is highlighted and an attribute is specified on a QBE template and then all tables having the attribute are searched for the value specified. When the attribute value is not specified, all tables are "grep"-ed to find the specific values regardless of which attributes they are stored in.
f) In all queries, we would like to ban the "building part" of the join operators. The relational join (in all forms and shapes –equi, theta, inner, outer, cross product) is a syntactic operator that creates new data objects that are not "semantically correct". We prefer the result of each query involving a join to be the set of all the intermediate results as separate relations - sub-relations of the arguments – and which inherit their semantics from the basic relations and thus are semantically correct. These subsets are obtained using the same filtering and are ordered in the same way it is done in the relational algebra (using the join-clauses and restriction-clauses) but the result of such query is a set of subrelations rather than a single relation. We insist on this change because we store the results of the queries in the database, and undefined semantics of syntactic objects will lead to miss-interpretations[8]. Aligning of the corresponding tuples of the sub-relations in a way similar to the end-result of the join and the requested ordering is useful.
g) Schema modification and data translation. As new data arrives, the schema needs to be modified to accommodate the changes. Operators that allow the change of the table schema such as adding a new column, changing the names of some attributes, and sometimes even the name of the table. What makes it to be the same table is the primary key that identifies the table regardless whether the table corresponds to an entity or a relationship. As long as the primary key remains unchanged, the table is the same regardless of its name and attribute name changes. If the primary key is modified, then a new table is created and the old one along with its data remains unchanged. A way to implement these changes is by creating a "surrogate" in the system composed from the values of the primary key attributes. This PK surrogate uniquely identifies the table and therefore its name can be modified without affecting

[8] An example with two relations: student-enrollment(StudentID,CourseID,Semester, Grade) and DriversLicense(UserID,UserName,Operator_Class). One can join the two relations where Grade=Operator_Class which will join together each of the C students with every driver with Operators Class C! Storing such a result can be miss-interpreted.

the internal representation. Changes that do not affect the primary key can be introduced directly to the table, e.g. a new column which would only be populated from follow up data entries. Older tuples would remain partially empty unless the user explicitly enters new values. Some of the older values may be reformatted as long as they are not part of the primary key. Since the primary key is maintained, partially filled tuples can coexist with fully populated ones.

The above list is by no means complete. A lot more engineering work needs to be done to make it usable.

4.2 The Concept of Sibdata

In the SD environment some of the ACID properties become irrelevant. For example, the user has no control over some of the remote sites. However, durability is of great concern not only on the server site but also on the local site. For example, in a travel arrangement where several services are used to comprise a complete travel, the transactions with each of the services involved are limited to each individual service and no high level of transaction atomicity can be achieved across these services. And yet, the user is concerned with the durability of the data of all these interactions as they were produced for a single purpose. Because the ability to collectively or partially abort or roll back is not available across multiple web service providers, the user would have to devise compensating actions (transactions) to undo some of these transactions and *manually* handle the data involved. This task becomes difficult and tedious because the user has to dig into his cached data and extract all the pieces needed to do so. In some cases, the cache may be on the URL field of the browser or its temporary store, or in the user interface buffers that may be lost accidentally or forgotten during long interactions. What is needed is a facility for naming and storing all the data generated during the user's interactions with multiple services, and an aggregation mechanism to tie "*sibling data*" together into a single identifiable and referenced unit.

We introduce a data grouping mechanism called "***sibdata***". This is an analogous to the concept of the transaction, which groups and treats a collection of actions as a single processing unit on the database substrate. The sibdata is an aggregation of all the data produced by one or more transactions and which data may or may not have logical relationships other than the fact they were "*siblings*"; that is they were generated as part of high level activity or inter-related activities, and therefore, all its pieces are dependent on each other. In SQL jargon, this would be "group by affiliation" rather than "group by value". Sibdata is a defining and naming mechanism. The grouping is on the data not on the transactions involved, and, thus, it is not a high-level transaction. However, each sibdata is atomic in the sense that the individual data glued together to compose a sibdata have no significance individually. Naming, saving into persistent storage, accessing, and maintenance of sibdata become a user's responsibility and provide him/her a Conceptual Modeling tool for capturing and managing his/her data service interactions. Sibdata is a live encapsulation mechanism that allows the user to create his own data aggregates. It is a convenient and easy to manage data scratchpad. It is also light-weight because it consists of metadata only and can be easily moved to and/or accessed from remote sites.

Sibdata is a concept that is missing from the database world. The reason for this may have been that the capturing of raw data, the loading of it in the database, and all the administration of these processes were considered being, like schema definition, centralized database activities performed by the DBA. The Extract-Translate-Load (ETL) activities were hidden from the users. Sibdata may bear some similarity to Object Oriented database modeling, in which objects were capturing hierarchically aggregated data objects. However, objects in an OO database model encapsulate access paths imposed by the hierarchy and thus inherit data dependence. In contrast, sibdata does not impose any data dependent access paths but treats the components as a set. No additional semantics are implied other than the subcomponents are glued together.

In the following, we define some basic primitives for sibdata. In a follow up paper we will define the full set of semantics for distributed sibdata and operational semantics for storage management and logging, support for Service Level Agreement, etc.

Sibdata Primitives

a) ***Naming:*** A sibdata obtains a unique name identifier by the primitive operation *begin_sibdata* <sibdata_name>. This primitive starts the collection of all data interactions under the chosen <sibdata_name>. The sibdata remains open until an *end_sibdata* <sibdata_name> that closes the sibdata. The data captured under the sibdata includes not only the data directed to the screen but also to other output ports.

b) ***Annotating:*** A sibdata is annotated by the user, and such annotations become meta-data used for searching within a sibdata or in a database of sibdata. Annotations reference data in sibdata and these references are captured (tagged) within the sibdata. User annotation is a process that continues after a sibdata has been closed by the user. The ability for the meta-schema to evolve is needed as the user discovers new and better ways to organize and search the sibdata. Tagging is a form of annotation, and is used to relate a metadata name/label or an annotation to a specific position on the sibdata. The user tags are used to search and access sibdata. All annotations of a sibdata are deleted when the sibdata is dropped.

c) ***Aggregation/Composition:*** Sibdata may be aggregated/composed with other sibdata to form new sibdata: *Create_sibdata* <sibdata_name> *with* <sibdata_name> [,<sibdata_name>]. The default semantics would be a light-weight grouping where the components are referenced only, but deep copy can be useful in some cases. A sibdata can be also disaggregated or decomposed by dropping it, *Drop_sibdata* <sibdata_name>. The sibdata will then be dropped if it is not referenced by other sibdata. A reference count index has to be maintained to avoid dangling references with the drop of a sibdata.

d) ***Maintenance and Scripting Control:*** Management and archival semantics are needed. Some of these can be done through some scripting language. These scripts would be used to display, align or synchronize received data but do not control the data generation. The scripts allow one to write data driven management scripts for the sibdata. These control primitives deal only the data flow and the internal arrangement or positioning of the data components and not with the control and access paths of the data sources. Therefore, they are data independent.

e) ***Logging:*** Sibdata will also capture the logs of the interactions of services. This includes time of invocation, parameters, etc. Assuming that a local log is maintained, each sibdata will store a pointer to the beginning of the position of the log recording the begin_sibdata and a pointer to the record at the end of it.

The semantics of sibdata are now developed for the distributed environment. Issues of trust, compliance of Service Level Agreements and dispute resolutions are inherent to the web service environment.

5 Conclusions

In this paper we advocate for the need to bridge the gap between schema design and database operation. The schema-during approach allows the user to perform CM during the database operation and provides a continuum for these activities. We also introduced the concept of sibdata for aggregating and managing sibling data.

The users need to be trained and equipped with high-level tools to operate in a schema-during approach. These tools will allow the users to do conceptual modeling (schema definition and modification) and data management (loading, accessing, and archiving). We believe that managing the evolution of data and training the end-user to cope with it is the only viable solution to the explosive data production of the digital revolution and the constant change of the world we live in.

Acknowledgement

We like to thank Christoph Moser for his help on the coverage of CM and the suggestions of the editor Alex Borgida on the paper organization. We also acknowledge U2 for the conception of these ideas.

References

[Ab74] Abrial, J.-R.: Data Semantics. In: Klimbie, J.W., Koffman, K.L. (eds.) Data Management Systems, pp. 1–59. North-Holland, Amsterdam (1974)

[AG08] Angles, R., Guteierrez, C.: Survey on graph database models. Computing Surveys (CSUR) 40(1) (February 2008)

[Al00] Alencar, F.M.R., Castro, J., Filho, A.C., Mylopoulos, J.: From Early Requirements Modeled by the i* Technique to Later Requirements modeled in Precise UML. In: WER 2000, pp. 92–108 (2000)

[BLM92] Batini, C., Lenzerini, M., Navathe, S.: Database Design: An Entity-Relationship Approach. Benjamin Cummings Publishing (1992)

[BMF79] Breutmann, B., Mauer, R., Falkenberg, E.: CSL: A language for defining conceptual schema. Elsevier, North Holland 1 (1979)

[BMS84] Brodie, M., Mylopoulos, J., Schmidt, J.: On conceptual modelling: perspectives from artificial intelligence, databases, and programming languages. Springer, New York (1984)

[BRJ05] Booch, G., Rumbaugh, J., Jacobson, I.: The Unified Modeling Language User Guide. Addison-Wesley, Reading (2005)

[Ch76] Chen, P.: The entity-relationship model: Towards a unified view of data. ACM Transactions on Database Systems 1(1) (1976)

[CKM00] Castro, J., Kolp, M., Mylopoulos, J.: Developing Agent-Oriented Information Systems for the Enterprise. In: Proceedings Second International Conference on Enterprise Information Systems, Stafford, UK (July 2000)

[Co70] Codd, E.: A relational model for large shared data banks. Communications of the ACM 13(6), 377–387 (1970)

[Co79] Codd, E.: Extending the database relational model to capture more meaning. ACM Transactions on Database Systems 4 (1979)

[De79] De Marco, T.: Structured Analysis and System Specification. Prentice-Hall, Englewood Cliffs (1979)

[DH72] Dahl, O.-J., Hoare, C.: Hierarchical program structures. In: Dahl, O.-J., Dijkstra, E., Hoare, C. (eds.) Structured Programming. Academic Press, London (1972)

[EW05] Evermann, J., Wand, Y.: Toward Formalizing Domain Modeling Semantics in Language Syntax. IEEE Transactions on Software Engineering 31(1), 21–37 (2005)

[Fe02] Fensel, D., Hendler, J., Lieberman, H., Wahlster, W.: Semantic Web Technology. MIT Press, Boston (2002)

[Fr06] Franklin, M.: The Structure of (Computer) Scientific Revolutions. In: Keynote at Dow Jones Enterprise Ventures, San Jose, CA (2006)

[HK87] Hull, R., King, R.: Semantic Database Modelling: Survey, Applications and Research Issues. ACM Computing Surveys 19(3) (September 1987)

[HM78] Hammer, M., McLeod, D.: The semantic data model: A modelling mechanism for database applications. In: Proceedings of the 1978 ACM SIGMOD international conference on management of data, pp. 26–36. ACM, New York (1978)

[HSA04] Hussain, T., Shamail, S., Awais, M.M.: Improving quality in conceptual modelling. In: OOPSLA 2004: Companion to the 19th annual ACM SGPLAN conference on Object-oriented programming systems, languages and applications, October 2004, pp. 171–172. ACM, New York (2004)

[Ju08] Juhrisch, M.: Using Enterprise Models to Configure Service-oriented Architectures. In: Tagungsband – MKWI 2008 (2008)

[Ka08] Kaschek, R.: On the evolution of conceptual modelling. In: Dagstuhl Seminar Proceedings 08181, Wellington, New Zealand, `http://drops.dagstuhl.de/opus/volltexte/2008/`1598 (access, December 2008)

[MS02] Maximilien, E.M., Singh, M.P.: Conceptual Model of Web Service Reputation. ACM SIGMOD Record 31(4) (December 2002)

[My78] Mylopoulos, J., Bernstein, P.A., Wong, H.K.T.: A language facility for designing interactive database-intensive applications. In: SIGMOD 1978: Proceedings of the 1978 ACM SIGMOD international conference on management of data (1978)

[My90] Mylopoulos, J., Norgida, A., Jarke, M., Koubarakis, M.: Telos: Representing Knowledge About Informatin Systems. ACM Transactions on Information Systems 8(4), 325–362 (1990)

[My92] Mylopoulos, J.: Conceptual Modelling and Telos. In: Loucopoulos, P., Zicari, R. (eds.) Conceptual Modelling, Databases and CASE: An Integrated View of Information Systems Development. McGraw Hill, New York (1992)

[My98] Mylopoulos, J.: Information Modeling in the Time of the Revolution. In: 9th International Conference on advanced information systems engineering (CA ISE 1997), May 1998, vol. 23(3-4), pp. 127–155. Elsevier Science Ltd., Amsterdam (1998)

[OMG07] OMG: OMG Unified Modeling Language (OMG UML) Infrastructure. Version 2.1.2, `http://www.omg.org/docs/formal/07-11-04`.pdf (access: December 2008)

[Pa72] Parnas, D.L.: A Technique for Software Module Specification with Examples. CACM 15(5), 330–336 (1972)

[Qu 68] Quillian, R.: Semantic memory. In: Minsky, M. (ed.) Semantic Information Processing, pp. 227–270. MIT Press, Cambridge (1968)

[RLD08] Roach, T., Low, G., D'Ambra, J.: CAPSICUM A Conceptual Model for Service Oriented Architecture. In: SERVICES 2008: Proceedings of the 2008 IEEE Congress on Services. IEEE Computer Society, Los Alamitos (2008)

[RM75] Roussopoulos, N., Mylopoulos, J.: Using semantic networks for database management. In: VLDB 1975: Proceeedings of the 1st International Conference on Very Large Data Bases (1975)

[Ro77] Ross, D.: Structured analysis: A language for communicating ideas. IEEE Transactions on Software Engineering 3(1), 16–34 (1977)

[Ro82] Roussopoulos, N.: The Logical Access Path Schema of a Database. IEEE Trans. Software Eng. 8(6), 563–573 (1982)

[RY84] Roussopoulos, N., Yeh, H.T.: An Adaptable Methodology for Database Design. ACM Transactions on Database Systems 17(5), 64–80 (1984)

[SS77] Smith, J.M., Smith, D.C.P.: Database Abstractions: Aggregation and Generalization. ACM Transactions on Database Systems 2(2) (June 1977)

[St02] Stolte, C., Tang, D., Hanrahan, P.: Polaris: A System for Query, Analysis and Visualization of Multi-dimensional Relational Databases (extended paper). IEEE Transactions on Visualization and Computer Graphics 8(1) (January 2002)

[Su05] Susi, A., Perini, A., Mylopoulos, J.: The Tropos Metamodel and its Use. Informatica 29, 401–408 (2005)

[TF76] Taylor, R.W., Frank, R.L.: CODASYL database management systems. ACM Computing Surveys (CSUR) 8(1), 67–103 (1976)

[TL76] Tsichritzis, D.C., Lochovsky, F.H.: Hierarchical database management: A survey. ACM Computing Surveys (CSUR) 8(1), 105–123 (1976)

[Ts76] Tsichritzis, D.C., Lochovsky, F.H.: Hierarchical database management: A survey. ACM Computing Surveys 8(1), 105–123 (1976)

[VdM05] Vara, J.M., de Castro, V., Marcos, E.: WSDL Automatic Generation from UML Models in a MDA Framework. In: Proceedings of the International Conference on Next Generation Web Services Practices (2005)

[Yu95] Yu, E.: Modelling Strategic Relationships for Process Reengineering. PhD thesis, University of Toronto, Department of Computer Science (1995)

[Zl77] Zloof Moshé, M.: Query-by-Example: A Data Base Language. IBM Systems Journal 16(4), 324–343 (1977)

On Conceptual Content Management
Interdisciplinary Insights beyond Computational Data

Joachim W. Schmidt

Institute for Software Systems
Hamburg University of Science and Technology
j.w.schmidt@tuhh.de

"Move away from any narrow interpretation of databases and expand its focus to the hard problems faced by broad visions of data, information, and knowledge management."

Motto of the 12th International Joint Conference on Extending Database Technology and Database Theory, Saint-Petersburg, 2009

Abstract. Instead of looking at content management as an extension of traditional database technology ("blobs") we exploit – in close cooperation with our colleagues from Art History – the notion of *symbol* as defined by Ernst Cassirer [1] in the cultural sciences. Since context binding, its definition, application and control, is crucial to cultural symbol management, and, therefore, to content and concepts as the two closely intertwined sides of cultural symbols, its demands are detailed and designed in a λ-calculus framework. Cassirer's request for *open* and *dynamic* symbol definition and modification leads us to a generator framework for the implementation of Conceptual Content Management Systems (CCMS). This presentation of our work is slightly allegoric and concentrates on the foundation, rationale and implications of this research.

1 Introduction: Motivation and Overview

When starting our research in conceptual content modeling and management, we considered two substantially different alternative paradigms to orient and direct our work:

1. Either we could follow a conservative approach and develop content management strictly out of Computer Science, by essentially "Extending Database Technology" ([2] etc.). In this case we would consider content management as a specific data-intensive and user-oriented application area with domain-specific models, languages and methodologies.
2. Or we could start from the humanities and - based on Marshall McLuhan and his general notion of media as "The Extensions of Man" [3], try to understand content management in terms of such extension and its roots. Based on these insights, user-adequate models for media content could be

developed along with model-based software architectures and systems for the extended functionality of (multi-) media content management.

Like many of our colleagues, in our earlier work we essentially took route number one, and tried to advance computer science by extending systems along their technical dimensions:

– transient data were generalized into objects with lifetime including persistence [4]
– built-in monomorphic types were extended into extensible polymorphic type systems [5]
– single-threaded programs were expanded to multi-threaded executions [6]
– centralized computers were integrated into communicating computer networks, etc.

The issue of "usability and responsiveness" usually was regarded as a secondary one: "appropriate" user interfaces were added to technical solutions, and technically interested and trained user communities learned how to cope with them.

There were essentially two lines of influence which led us to choose route number two for our work in conceptual content management.

First of all, there was, what I would call the Toronto influence: John Mylopoulos' early work on Conceptual Modeling [7] (and our marginal contributions to it) made us sensitive to the higher-levels of application modeling beyond the use of programming languages and database models [8]. In addition, there was the influence of Marshall McLuhan's work and of our occasional visits to his "Center for Culture and Technology" at the University of Toronto in the late 70ies. McLuhan's understanding of media as "the extensions of man" triggered our view of content as something substantially different from just being "blobs" - binary large objects.

In addition, we were influenced by our specific Hamburg constellation: in the early 90ies we had started an interdisciplinary research project with partners from Art History (Martin Warnke and his Hamburg institute). Due to their work on *political iconography* [9] we had access to ample image, text and data material organized around concepts from the domain of art and politics - although all this material was represented through paper cut and paste. Methodologically, Hamburg art historians are still heavily influenced by their former colleagues Aby Warburg and Ernst Panofsky, from whom we learned not only how to establish adequate *descriptions* of objects of art but also how to (partially) capture their *meaning*.

Finally, and most importantly, this "Warburg connection" was our bridge to Ernst Cassirer who also had worked at Hamburg University - he even was one of its presidents before he had to leave Germany in 1933 - and his profound research in *Symbolic Forms* [1, 10, 11]. According to Cassirer, man's most important capacity is his ability to freely create his own symbol systems, and the *modus* *operandi* of extensions of man is his capability to do *open* and *dynamic* symbolic form*ing* – man as *animal symbolicum* [1] and[1] as *symbolic species* [12].

Cassirer defines symbols as indivisible units capable "to encompass the totality of those phenomena in which the sensuous is in any way filled with meaning (... "SinnerfÃijllung" des Sinnlichen)", and what humans recognize is not just "a simple intuitive datum", instead it does "follow from a process of symbolic formation (... ist Ertrag und Ergebnis eines Prozesses der symbolischen Formung [1])." And this process of symbolic formation is *open*, no predefined and fixed ontologies, and *dynamic*, a continuous activity, highly personalized. This means that Cassirer leaves Kant's fixed framework of a closed, time-unrelated apriori in favor of open and dynamic aprioris legitimated pragmatically only by their performance and truth in processes of interest [11].

Based on the above mindset we structured our research programme on conceptual content management as follows. After consolidating our view on symbols along the line of Cassirer and McLuhan we modeled Objects of Art [13] (and other content-intensive artifacts) as content-concept pairs with a functional approach to the definition of bindings and to binding control (section 2). Consequently, when designing our languages for content definition and querying we started from a typed λ-calculus as a model basis for content abstraction and application (section 3). Since we see content as typed symbols with the binding abilities of linguistic objects [13] and since, due to Cassirer, symbols have to be defined and manipulated within *open* and *dynamic* environments we choose a generator technology for system implementation (section 4).

To summarize, the main purpose of this interdisciplinary work was to gain a deeper insight into content management based on a generalized notion of symbol. In addition, we wanted to benefit methodologically from the experience of Art History as a discipline with a long tradition in *understanding*, and not just *describing*, conceptually complex content. And, in doing so, hoping that this know-how transfer into Computer Science may result in some interesting scientific and technical insights in general content management which, perhaps, might lead to some commercially relevant results [14].

In our interdisciplinary cooperation project with our colleagues from Art History we, as trained natural scientists with a programming language and database background, learned the following message which essentially determined this research:

> "In Kant's expression, the natural sciences teach us "to spell out phenomena in order to read them off as experiences"; the science of culture teaches us to interpret symbols in order to decipher their hidden meaning, in order to make the life from which they originally emerged visible again." [10, p. 86]

[1] While for Cassirer [1] symbols are the basis for knowledgeable representations, Deacon [12] emphasizes the role of symbols for meaningful identification. Computer Science, finally, specializes in symbol manipulation, in particular, symbol reduction. This threefold use of symbols deserves, in our opinion, a deeper and coherent investigation.

2 Conceptual Content Modeling: Foundation and Rationale

Computer science was always regarded as a symbol manipulating discipline. The general notion of symbol is that of instance-category or token-type pairs, e.g., `<7, integer>` or `<true, Boolean>`.

In the following, we will briefly sketch Cassirer's view of symbols as general content-concept pairs and the role they play in the natural and in the cultural sciences, and relate this view to our work in computer-based conceptual content management.

Cassirer [10] accepts a notion of symbol similar to that of the natural sciences, but generalizes it from its limitations by theory-based categories and extends its usability to cultural sciences.

Formal categories turn out to be appropriate for representing the results (or their abstractions) of what Cassirer calls *perception of things* (*Dingwahrnehmung*). This form of perception may best be characterized by the relationship between the recipient and the received: in the natural sciences, the received is conceived by the human receptor as an *alien thing - aliena res, aliud*, the *me* and the *it.*

Cultural sciences, on the other hand, are characterized by what Cassirer calls *perception of expression* (*Ausdruckswahrnehmung*), the received is recognized by the human as an *alter ego*, the *me* and the *you.*

Cultural sciences are, and have to be, far less formal than the natural sciences; they characterize their categories not by axioms or formal denotations. Instead, cultural objects are characterized by a variety of different concepts of *form and style.*

> "The wealth of different concepts of form and style that the science of culture develop serves finally this one task: only through them is the revival, the "palingenesis", of culture possible." [10, p. 77]

How is meaning then assigned to cultural categories if not by formal statements? How else can this reflexive and analytic process which is so essential for the understanding of objects of art be supported? In particular, since, as Cassirer emphasizes

> "Culture is forever creating new linguistic, artistic, and religious symbols in an uninterrupted stream. However, science and philosophy must decompose the symbolic language into its elements in order to make them intelligible. They must treat analytically what was produced synthetically." [10, p. 86]

One pragmatic way for supporting this process of understanding as well as teaching and learning in Art History is realized by exemplifying categories by selected sets of particularly striking examples ("best practice") [15, 16, 9]. Cassirer also insists on emphasizing that the instance-category relationship in the natural sciences and in mathematics differs essentially from that in the cultural sciences. Type inference and type subsumption, for example, on the one hand, and the use of selected instances as training sets (e.g., [17]) on the other.

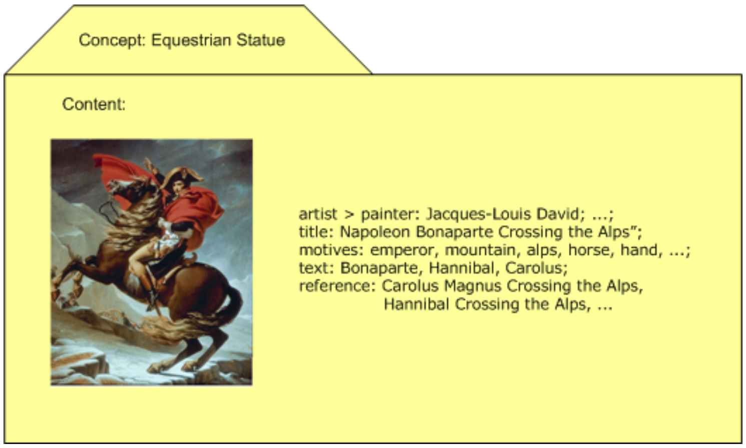

Concept: Equestrian Statue
Content:
artist > painter: Jacques-Louis David; ...;
title: Napoleon Bonaparte Crossing the Alps";
motives: emperor, mountain, alps, horse, hand, ...;
text: Bonaparte, Hannibal, Carolus;
reference: Carolus Magnus Crossing the Alps,
Hannibal Crossing the Alps, ...

Fig. 1. Image card: Equestrian statue [9]

"We understand a science in its logical structure only once we have clarified the manner in which it achieves the *subsumption of the particular under the universal*." [10, p. 69]

As an example for subsumption in the cultural sciences, Cassirer refers to Jacob Burkhard when he tried to specify the characteristics of the "Renaissance Man", of the leading personalities of the Quattro cento, of a Leonardo Bruni, Lorenzo Valle, . . . da Vinci, Machiavelli, Michelangelo, Borgia, etc. The result was that whatever set of characteristics was used for one person didn't fit most of the others.

"The particular individuals belong together, not because they are alike or resemble each other but because they cooperate in a common task, which, in contrast to the Middle Ages, we sense to be new and to be the distinctive "meaning" of the Renaissance. All genuine concepts of style in the culture of science reduce, when analyzed more precisely, to such concepts of meaning." [10, p. 72]

So, the instance-category relationship in the cultural sciences is usually not defined axiomatically by being a successor of zero or by being generated by sequences of push and pop operations. Instead, persons, for example, are qualified under the category "Renaissance Man", by meeting certain non-functional requirements such as working towards certain goals or sharing given standards or quality constraints [18, 19]. Despite these differences between natural and cultural sciences, Cassirer proposes for both areas the same notion of symbol, defined as indivisible pairs of the *sensuous content* and its *conceptual meaning* [1].

Warnke's project on Political Iconography (PI) gave us the unique opportunity to work with some hundred thousands of such symbols (Fig. 1, Equestrian statue) represented by image cards. They are essentially used to define the categories relevant to the domain of political iconography and to give them meaning by carefully selected example sets [9]. We computerized a subset of Warnke's material (see, http://www.welib.de/) essentially by following Cassirer's notion of symbol. For our purpose, we call such symbols *assets* [20].

Cassirer also recognized the importance of *persistence* for his generalized notion of symbols. Cassirer calls persistent symbols *monuments*:

> "What is actually preserved for us from the past are specific historical monuments, "monuments" in words and writing, in image and bronze. This becomes history for us only once we begin to see in these monuments symbols not only in which we recognize specific forms of life, but by virtue of which we are able to restore them for ourselves." [10, p.77]

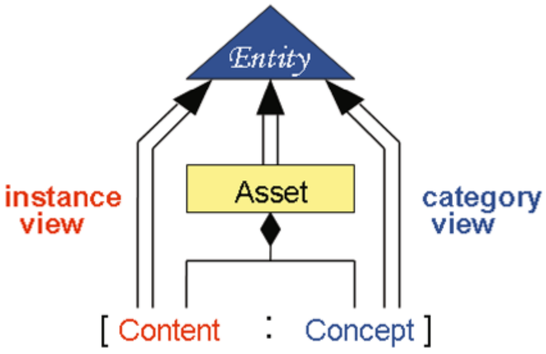


Fig. 2. Assets as content-concept pairs

Cassirer's model corresponds in a sense to what we had found essential for the realm of databases: content vs. values, concepts vs. types or domains, symbols vs. typed objects, monuments vs. persistent objects and, finally, there is the overall correspondence of goals: "to make monuments readable again" [10, p.77].

3 Languages for Conceptual Content: Design and Use

In his essays on *Art and its Objects* [13] Richard Wollheim also chooses a type-token-framework for discussing the essence of cultural entities. After rejecting several alternatives (such as the material object hypothesis, the Ideal theory and the Presentational theory) Wollheim favours a view of art objects as linguistic objects. Referring to Ludwig Wittgenstein [21], Nelson Goodman [22] and others, Wollheim emphasizes the analogy between art and language, since "both are heavily intertwined with the complex of our habits, experiences, skills" and that both cannot exist outside this relationship (and vice versa).

As a consequence, adequate representations of objects of art have to master, just as linguistic objects do, rather complex and variable binding schemes relating a wide variety of objects often reaching into domains outside art. Such bindings, which a learned art historian is making *implicitly*, have to be made *explicit* in any representation for it to be meaningful to others (and to computer systems).

Asset Scheme:

LET any BE λc:any_jpeg.c — /scheme for creating assets of concept "any" and content "any_jpeg" file

LET image BE REFINEMENT OF any — /scheme for creating assets of concept "image" and any jpeg content

LET *paradise* BE REFINEMENT OF image — /scheme for creating assets of concept "paradise" and "image" content

. . .

Asset Base:

LET C BE

: any_jpeg

LET A_0 BE (CREATE any)C — /asset of concept "any" and jpeg file content C (*Adam and Eve*; λ-application; Panofsky level 0)

LET A_0 BE (CREATE image)A_0 — /asset with content C specialized to concept "image" (still Panofsky level 0)

LET A_0 BE (CREATE paradise)A_0 — /asset with image *Adam and Eve* specialized to concept "paradise" (still Panofsky level 0)

LET A_1 BE (λm:material.A_0)canvass — /asset enriched by material description (lifted to Panofsky level 1)

LET A_2 BE (λt:temptation.A_1)snake — /asset enriched by iconic meaning of "snake" (lifted to Panofsky level 2)

Fig. 3. Asset refinement for Dürer's etching *Adam and Eve*

Computer scientists have learned to understand and master such binding demands essentially by the means of λ-calculus (e.g., [23]). λ-expressions are linguistic entities for which, by *abstraction*, selected code elements can be distinguished as being bindable, and then be explicitly bound by *application* of other λ-expressions [20, 24].

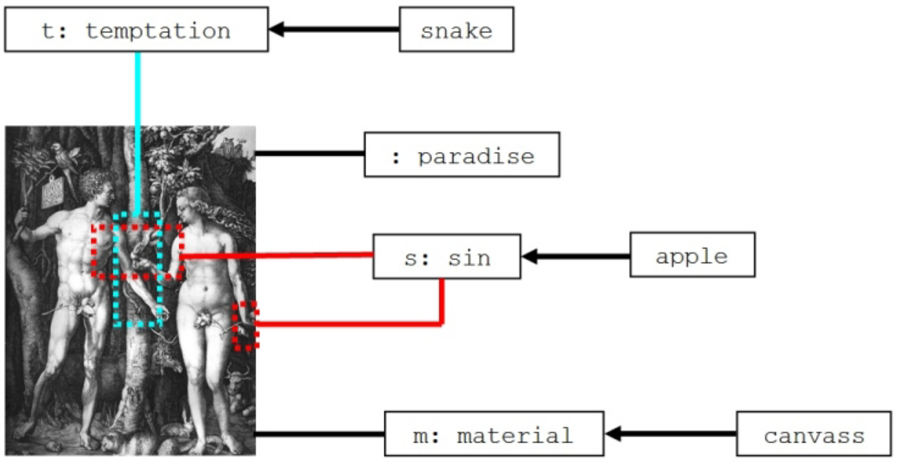


Fig. 4. Asset graph: abstraction, application and selection

Roughly speaking, by our asset approach we *treat content as code*[2] and allow typed asset abstraction and application over content (Fig. 3: Asset definition and refinement for Dürer's etching "Adam and Eve"). Consequently, the asset system has as its kernel a typed λ-calculus engine embedded into an o-o-like λ-expression editor, and sitting on top of an open and dynamic persistence machine (see Section 4). The asset system is accessible by a linguistic as well as a graphical interface.

Beginning with Aby Warburg [25] and in particular with Erwin Panofsky [26, 27, 28], art history also developed a methodological approach to the systematic specification of objects of art.

Based on the grounding level of pure *existence* (level 0) Panofsky introduces a first level, where *descriptions* (*Beschreibung*) of objects of art are given in terms of general purpose domains such as length, weight, time etc. or other generally understood terms.

Level 2 gives the *meaning* (*Bedeutung*) of objects by introducing the specific iconographic vocabulary used by the artists working in a certain domain. On this level, a snake, for example, is not just an animal, as a description given on level 1 would say; a snake on level 2 was for Dürer an icon for temptation or persuasion. Finally, on level 3, objects of art are additionally represented by general cultural *effects* (*Wirkung*) they cause or by which they are affected.

Panofsky (see table 1) emphasizes that higher strata include the lower ones, i.e., objects represented for example on the 3rd level of iconology also include a representation on levels 2, 1 and 0.

[2] Wollheim objects to the notion of content as code [13, p.83]. His understanding of code is, however, different of that of computer scientists: he refers to code as a specific encoding of a language, e.g., by semaphores, binaries or other alphabets and not as a piece of language text (as computer scientists do).

Table 1. Panofsky Strata; Example: Dürer's Etching *Adam and Eve*

Representation levels:

0. Individual (*existence* level): plain media view (image, text, speech, ...)
1. Characteristics (*description* level, pre-iconographic): color, sizes, age, ...; names (artists, periods, regions, ...) humans, animals, fruits, trees, ...
2. Iconography (*meaning* level): paradise, seducing Eve, curious Adam, tempting apple, sinful snake, ...
3. Iconology (*effect* level): Jewish and Christian ethics and legal systems, their origins and consequences, ...

If we compare this approach with computational data modeling then *traditional databases* would best correspond to Panofsky level 1. Application entities are described by reference to general-purpose domains such as space, time, money etc. If such descriptions are highly regular and, in addition, content is *empty* then assets collapse with the schema-defined Cartesian structures of relational tuples or with class-generated objects. In this sense, Panofsky level 2 could best be compared with the domain-specific approach of (*digital*) *libraries* [29] [30] while Panofsky level 3, allowing general reference to cultural concepts and content, might best be related to open *web-provided content* [31].

Panofsky's methodological approach to image description and understanding was not unanimously accepted. Critics were, for example, based on the fact that there was no room for any elements of *stile* and *form* used by an artist [32]. Panofsky did, for example, not model the expressive means of light or the specific use of certain colors, of shade, perspective etc. or the meaning of neighbourhood relationships. This omission of *expressional* properties leads to the *problem of integration and disintegration* of the iconology-oriented method [33] and is definitely a serious deficit in Panofsky's approach. This is particularly true since Cassirer has already emphasized (see above) that the *perception of expression* (*Audruckswahrnehmung*) is of central importance to the cultural sciences.

There are attempts to overcome such deficits, for example, for literary objects of art or linguistic content in general. Wilhelm Schneider [34], for example, positions literary objects in an n-dimensional space of ordered expression types (Ausdruckstypen) such as

– strength: weak $\leftrightarrow$ extensive
– agressivity: low $\leftrightarrow$ strong
– trust: little $\leftrightarrow$ substantial
– emotion: low $\leftrightarrow$ high

The meaning of such expression types are defined - just as Warnke does it for Political Iconography - by typical example sets from the literature.

Pre-iconographic, description-level bindings usually characterize an object as a whole (unless the object itself is a composit one, as, for example, a triptych). However, bindings to iconographic elements of images (Panofsky level 2) typically don't refer to entire images but only to selected elements. Therefore, *selector*

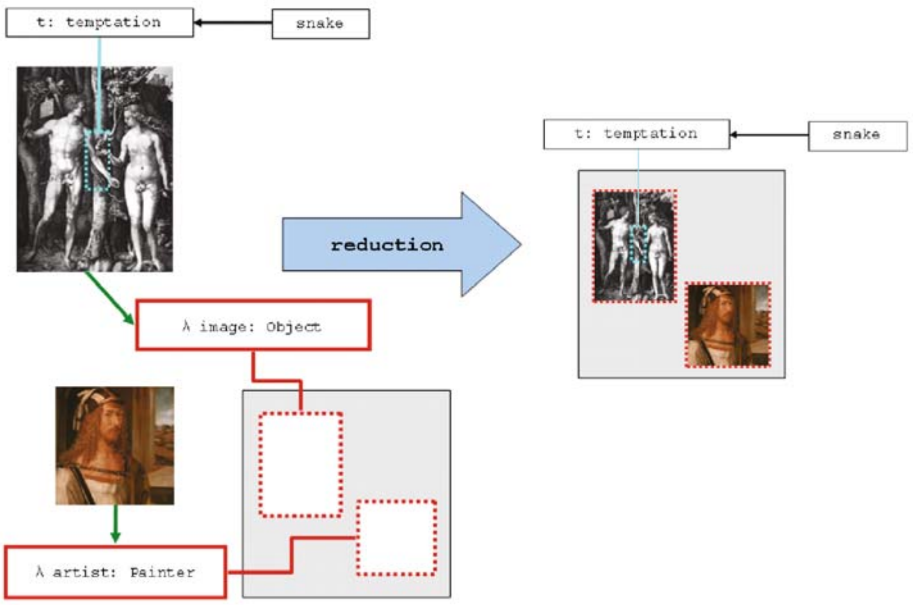


Fig. 5. Asset selection and reduction

schemes typical for assets of a certain kind, e.g., for 2-dimensional images, are made available. Spatial selectors can be applied to images and then, in a sense, be reduced to assets with spatially selected content (see Fig. 5: Asset selection and reduction).

The iconic elements of Dürer's "Adam and Eve", snake or apple, for example, can first be selected by means of some parameterized selector *spatial*; in a second step, λ-abstractions and λ-applications can be made: the iconographic definition of snake, for example, can be applied to the selected component instead of working on an asset as a whole (Fig. 4: Asset graph: abstraction, application and selection). Selection can be nested to any depth.

$$\mathtt{A}_2 = \lambda t : \mathtt{temptation}((\lambda s : \mathtt{spatial.A}_2)\mathtt{<4,4;6,7>})\mathtt{snake}$$

Only in rather specific cases can our λ-expressions be reduced algorithmically. Asset selection is one of these cases (Fig. 5: Asset selection and reduction).

In general, assets serve the purpose of being explicit persistent denotations of semantically rich entities represented by complex symbol networks. If at all we can see the process of "reduction" taking place in the mind of a human viewer, who, based on rich, informative and comprehensible binding structures, is able to "reduce" assets to its own mental objects of understanding. Therefore, the issue of graphical interface design and development was essential for this work (Fig. 6: Graphical user interface for asset manipulation and comprehension) [24].

In addition to system support for easy construction, access and comprehension of asset networks, our research also addresses issues of query languages for such

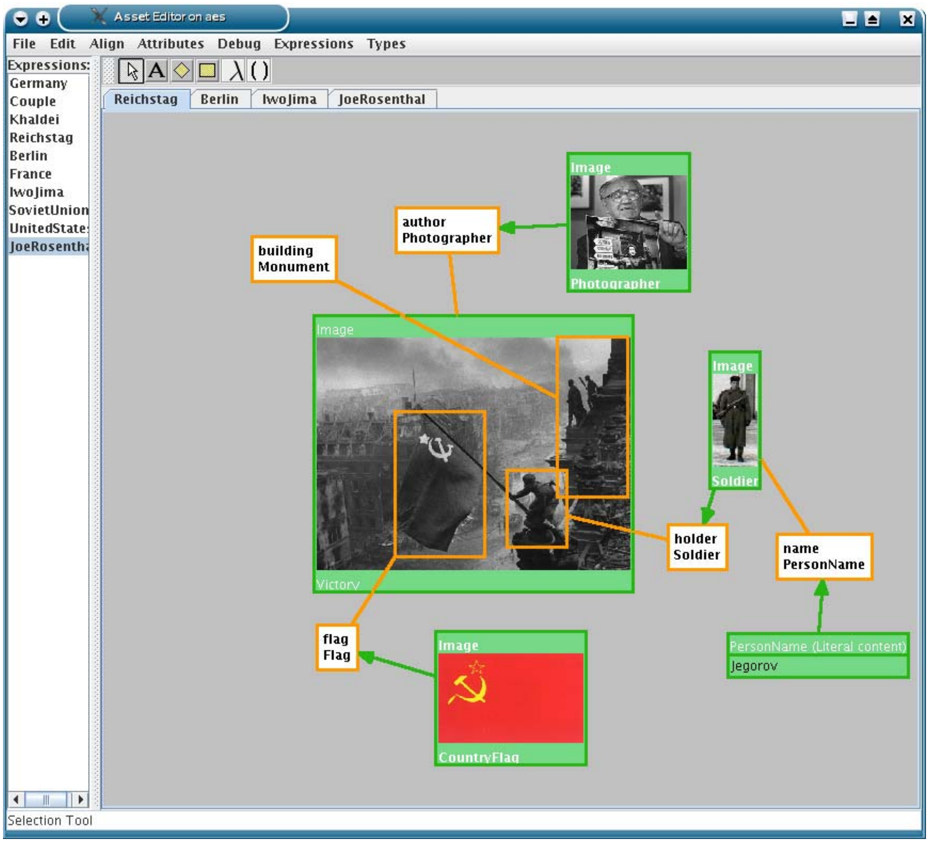
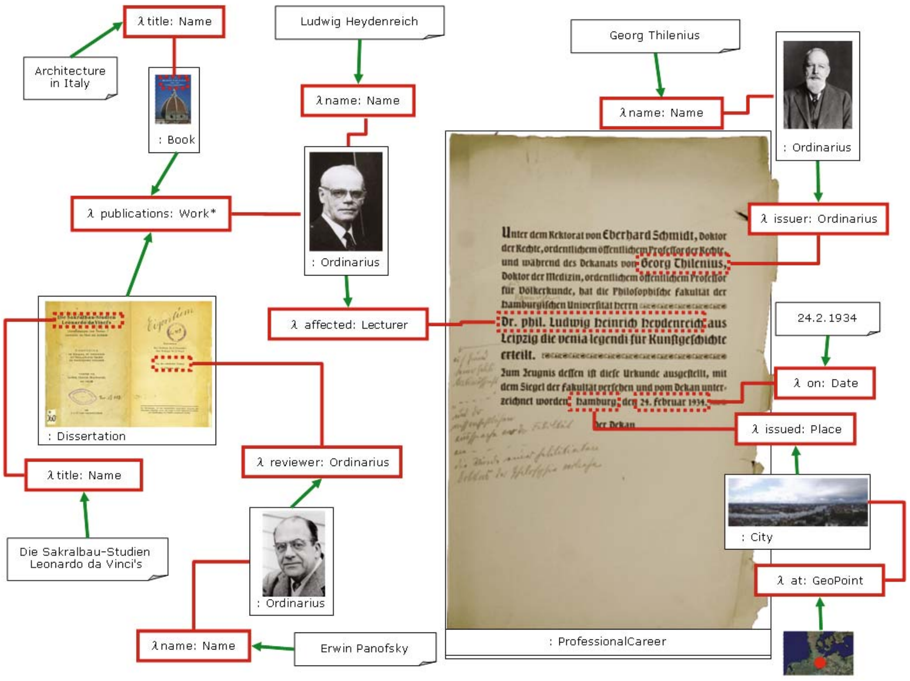


Fig. 6. Graphical user interface for asset manipulation and comprehension [24]

persistent asset systems. Query optimization is approached by exploiting partial regularities of asset graphs and mapping them into schemata which can finally be handled by conventional database technology. The syntactic and semantic fine print of the Asset Language and the system implementation can be found in [24, 20, 35].

We used and evaluated the asset language and its technology in extensive content-intensive applications. In the GKNS project, for example (GKNS: Geschichte der Kunstgeschichte in der Zeit des Nationalsozialismus; Art History in the Third Reich [36]), the Art History departments of Humboldt University, Berlin, the Universities of Bonn, Hamburg, Munich cooperated together with our institute at TU Hamburg as technology provider. The GKNS project had access to several large historical archives and evaluated extensive volumes of mainly administrative material on German art history departments during the time between 1930 and 1950. The project established a web portal giving access to a coherent network of resources (see, `http://welib.de/gkns`) for an intensive research on this topic. Since the portal intends do provide more than



Unter dem Rektorat von Eberhard Schmidt, Doktor der Rechte, ordentlichem öffentlichem Profeſſor der Rechte, und während des Dekanats von Georg Thilenius, Doktor der Medizin, ordentlichem öffentlichem Profeſſor für Völkerkunde, hat die Philoſophiſche Fakultät der hamburgiſchen Univerſität herrn Dr. phil. Ludwig Heinrich Heydenreich aus Leipzig die venia legendi für Kunſtgeſchichte erteilt.

Zum Zeugnis deſſen iſt dieſe Urkunde ausgeſtellt, mit dem Siegel der Fakultät verſehen und vom Dekan unterzeichnet worden. Hamburg, den 24. Februar 1934.

Der Dekan

Fig. 7. *Venia Legendi* for Ludwig Heidenreich: document and its context [24]

just large volumes of scanned documents, content as well as its context has to be modelled conceptually (Fig. 7: *Venia Legendi* for Ludwig Heidenreich: document and its context). The context includes assets on colleagues, administrations, publications, locations and institutions etc. and is intended to be as unbiased as possible. This *neutrality* of concepts and content is further encouraged by allowing a variety of personalized extensions and versions of asset networks [24].

Conceptually we position our work, as we have seen, in a language framework. Another reason for doing so is Cassirer's explicit demand for an *open* and *dynamic* approach to symbolic form*ing*. This implies, as discussed in the subsequent section, substantial technical consequences for a CCM system architecture. (Another reason may be our personal background in and preferences for computer languages and language technology).

4 A Generator Framework for Conceptual Content Management

It is central to the approach of Cassirer that the process of symbol forming has to be *open* and *dynamic*. If we are still willing to take Marshall McLuhan's view and accept a media approach to symbolic forming – media as extensions of man – then any extended support for the human symbol determination effort

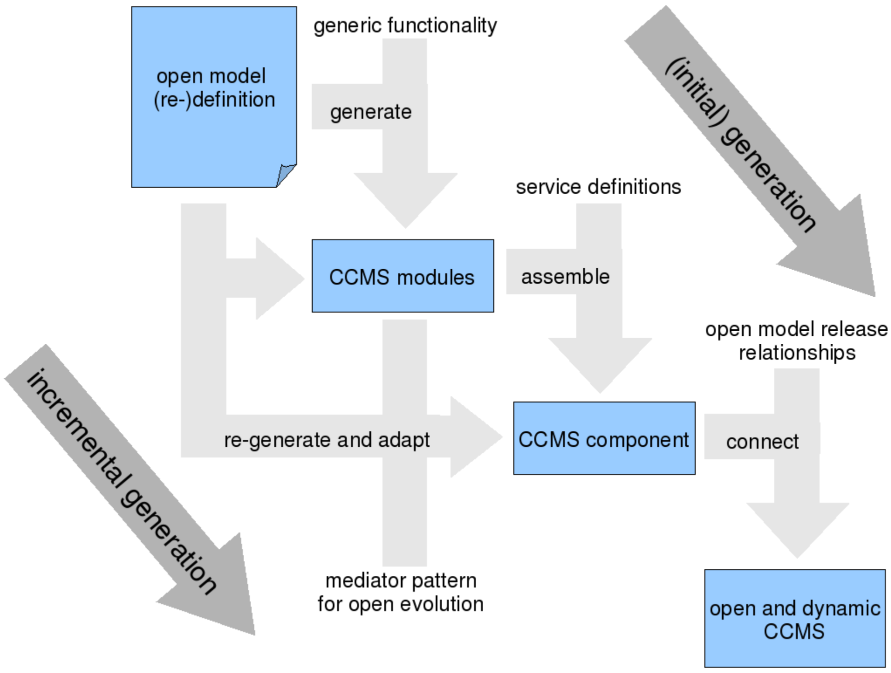


Fig. 8. CCMS generation processes

(*Bestimmungsleistung*), any symbol handling support system (and that's how we look at conceptual content management), also has to be open and dynamic. And open and dynamic modeling requires that domain modeling becomes part of the domain experts' working process.

For our assets, *openness* implies that any kind of a fixed, a-priori model is not appropriate. The choice of concepts to be assigned to content is totally free and not at all restricted by pre-defined and prescribed ontologies. Also, the exemplar sets (learning sets) used for concept definition are subject to changes. Such openness is also required for the refining and value-adding processes of content abstraction and application. Furthermore, for each existing asset such definitions and assignments must be changable unrestrictedly and at any time. While open model definition and change is a rather challenging implementation demand, the operational requirements of CCMSs are less demanding: it is generally sufficient that a generic CRUD-functionality (Create-Retrieve-Update-Delete) together with generic presentation as well as query and communication services be provided on openly defined symbols systems.

Dynamics means that the open flexibility sketched above has to be available "online" without any delay by any software engineering life cycles. Whenever a user wants to make changes, the CCMS has to react instantaneously. This also implies that all the generic services provided by a CCMS such as query facilities, presentation services, and in particular, all the communication and cooperation

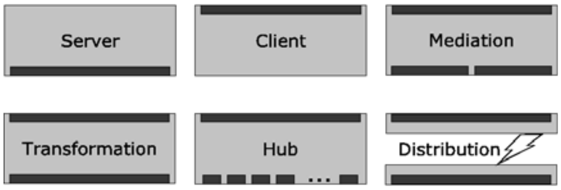


Fig. 9. Six kinds of CCMS modules

capabilities between sequences of model versions of a specific user as well as between the models of a cooperating user group have to work continuously under such changes.

Under the above conditions, traditional software engineering approaches such as generic software systems, domain-specific software, individually developed software, etc. turn out to be either too restricted – when it comes to model extension – or too powerful for the functional demands of CCMS operation.

As a consequence, we favour a *generative* approach to CCMS construction [37, 35]. Over a fixed basis of (essentially CRUD) functionality we generate a member of a family of CCM systems for each substantially changed version of asset models, including the communication glue to make such versions cooperate.

Figure 8 gives an overview of the CCMS implementation (i.e., generation) process. The arrows pointing downwards illustrate the initial generation of a CCMS, the arrows pointing from left to right the incremental generation that copes with model changes.

Generation starts from asset expressions in a language that allows to openly define, change, and interrelate domain models.

Implementations of generic functionality are generated in the form of so-called *modules*. Modules encapsulate one implementation of the generic functionality following the principle of *separation of concerns*. The kinds of modules for the most frequently occurring tasks are illustrated in Fig. 9. All modules have a uniform interface and can be composed in layers. This makes it possible to always combine modules in the way most appropriate to the task at hand. The module interface reflects generic functionality. Each module can thus base its functionality on calls to the module(s) of the underlying layer.

Modules are used to implement *components*. Each component encapsulates one CCM model as well as all the asset instances already created according to that model. A component offers services to the overall CCMS, where the services' functionality is assembled from the functionality provided by modules.

Therefore, modules are concrete units that establish code reuse, whereas components are logical units that establish content reuse.

Modules can be of several kinds:

– *server modules* access components via standard protocols.
– The data bound to asset instances are stored in third party systems, databases in most cases. Mapping asset models to schemata of such systems is done by *client modules*.

- A central building block of the architecture of generated CCMSs is the mediator architecture [38]. In our approach it is implemented by modules of two kinds. The first are *mediation modules* which delegate requests to other modules based on the request.
- The other kind are *transformation modules*. By encapsulating mappings in such modules rather than integrating this functionality into other modules, mappings can be added dynamically (compare [39]).
- *Hub modules* uniformly distribute calls to a larger number of underlying modules.
- By use of *distribution modules* components can reside at different physical locations and communicate by exchanging asset encoding in some marshalling format.

These module kinds have been identified with respect to the requirements of open and dynamic CCMSs. Various patterns are applied in order to offer specific CCMS services. For example, schema evolution leads to a combination of client, transformation, and mediation modules (indicated in Figure 10, see [20] for details).

The generation of various software artifacts is performed by a CCM model compiler. More precisely, we have defined a *compiler framework* that allows to define compiler instances for different CCMS setups. One of the extension points of this frameworks is the back-end, where different generators can be registered, each of which provides one implementation of one module kind.

One of the tasks of the compiler framework itself is the scheduling of generator execution. Schedules are computed based on information on producer/consumer relationships between the generators. Figure 11 illustrates this for the example of the generation of a web service server module. A set of generators is involved in this effort: the API generator generates the uniform API from the given model.

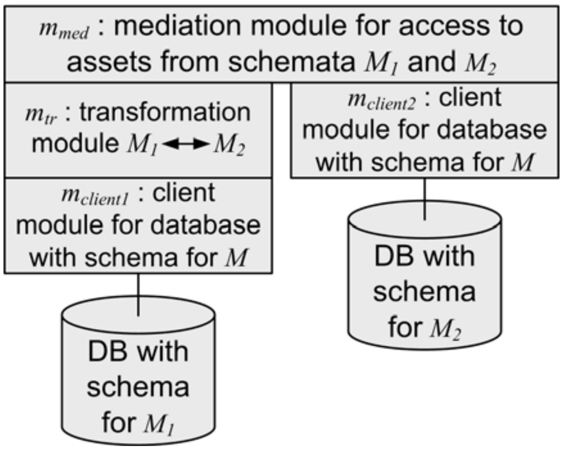

m_{med} : mediation module for access to assets from schemata M_1 and M_2
m_{tr} : transformation module $M_1 \leftrightarrow M_2$
$m_{client2}$: client module for database with schema for M
$m_{client1}$: client module for database with schema for M
DB with schema for M_2
DB with schema for M_1

Fig. 10. A sample module organisation

An XSD generator generates an XML schema definition that allows to encode assets of the given model in XML. A WSDL generator generates the web service descriptions for the services to be offered. Finally, the generator *WS Impl* creates the server module for Web Services. As shown in Fig. 11, *WS Impl* depends on the API (since this API has to be implemented) and the WSDL definition, which in turn depends on the XSD. Therefore, the generator executions have to be scheduled accordingly.

To summarize, the compiler framework in conjunction with the target architecture allows one to define software components with a functionality that is assembled from generic functionality. Compiler applications generate the required functions for assets according to a current model. Evolution is handled by incremental generation: a new CCMS is generated for a new model, an existing CCMS for the previous model is integrated. Content is retained and incrementally transformed to the new model. Communication between systems of different users or user groups is maintained in the same way.

The generative approach allows to meet the openness and dynamics requirements to a substantial degree: models can openly be defined and changed, and CCMSs follow such changes dynamically.

Open model changes are applied by defining a new model and relating it to the model currently in use. As seen above, asset redefinitions result in the generation of a new CCMS that starts a new content base with new schemata etc. This is why more changes are possible in model redefinitions than there are legal additions to, for example, subclass definitions in object-oriented models. While removing attributes from a subclass is not sound in an object-oriented type system, attributes can simply be dropped during asset model evolution and system generation – access to existing instances simply hides the attribute. For the same reason, extensional definitions of concepts can be changed.

The fact that existing CCMSs are kept as subsystems (in the form of components) during evolution (see Fig. 10) allows us to maintain working relationships between revisions and variants of domain models, each of which is implemented by a component of a CCMS. Content is preserved in the previous CCMS. Since schemata, formats, etc. remain unchanged, there is no need for instantaneous content migration. Instead, existing content can be accessed in terms of a new model and is incrementally migrated through mediators. The fact that this happens whenever a user works with an asset provides quality assurance, since there are only explicit updates, no automatic migration.

Since both the (generic) functionality as well as the services defined on top of that functionality stay the same for all CCMS revisions and are applied in a model-specific way, all other CCMS components as well as the CCMS users can continuously interact with the respective CCMS. This is how communication between users whose models have been derived from a common ancestor model is maintained. In the same way that users can access previously generated CCMSs as subsystems, communication uses adaptation and mediation to maintain communication paths between different CCMS components that include modules to access each other.

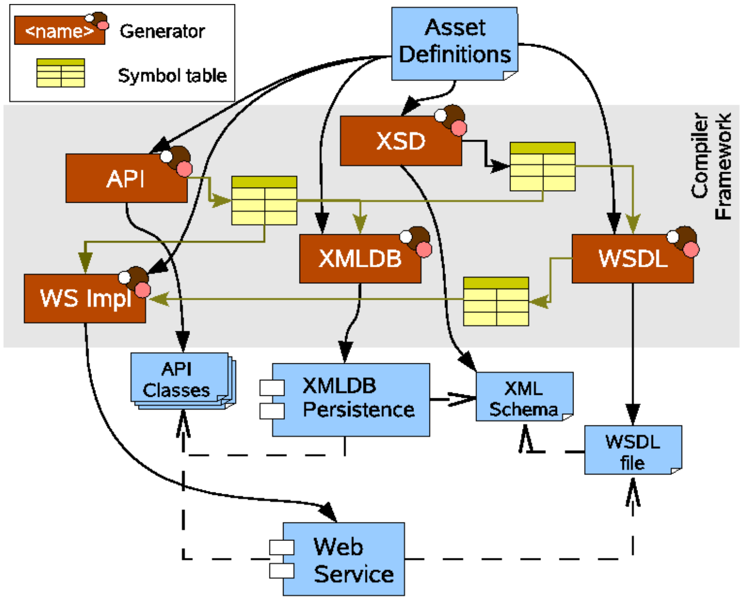


Fig. 11. Compiler framework

The ability to dynamically generate modules, to assemble them to form components, and to integrate an existing CCMS to form an updated CCMS is the means to achieve the desired dynamics. CCMS evolution does not interrupt the users' work since it does not involve manual software development cycles, and both the content base as well as existing communication paths are automatically maintained.

In section 2 we mentioned Cassirer's two different kinds of symbols, the ones in the cultural sciences essential for human communication and the ones based on the formal categories of the natural sciences which includes, of course, our discipline and its computational technology (*alter ego* vs. *aliena res*). From that point of view, our effort of CCMS software implementation is essentially a mapping exercise between these two kinds of symbols – the symbols in the cultural and in the natural sciences.

5 Conclusion

In this research we argue forcefully against any fixed and prescribed ontologies for conceptual content modeling and management. Instead, we follow Cassirer's approach according to which the essential capacity of man – of the *animal symbolicum* – is his abstract gift of open and dynamic forming of extensible symbol systems.

Cassirer's open and dynamic approach to the genesis and lifetime of symbols serves as the blueprint for our software system development in conceptual content management. Starting from simple services for computational content containers, such as jpeg or ASCii files or databases, content can be introduced into media objects and lifted to the objects of our *asset* language and system. Assets can then meander through the value-adding Panofsky strata of content description and meaning assignment: from the pre-iconic conceptual level with extensive object descriptions up to assets with an iconographic meaning and, finally, to the levels of iconological significance.

In the light of McLuhan, with his already very broad view on the notion of media (his view includes radio and television as well as cars and money as media) Cassirer's *symbolic forming*, i.e., the abstract capacity of humans to open and dynamically form extensible symbol systems should then be "the mother of all media" – the ultimate "extension of man".

And it is Computer and Communication Science which provides the means to substantially and adequately support such extensible symbol systems and symbolic media in multiple directions. Probably, it is this potential of our discipline and this need for our discipline which explains its surprising success and makes it *the* symbol-oriented medium.

Acknowledgement

I would very much like to give to my friends and colleagues Martin Warnke, Hans-Werner Sehring and Sebastian Bossung my sincere thanks for their constant interest in this research and for their substantial scientific and technical contributions. Many thanks also to Mrs. Helen M. Pert who looked, as she had always done in the past, over the English text.

References

[1] Cassirer, E.: Philosophie der symbolischen Formen, Hamburg (1923); see also: Felix Meiner Verlag, Hamburg (2001); The Philosophy of Symbolic Forms. Yale University Press, (1955)
[2] Schmidt, J.W., Missikoff, M., Ceri, S. (eds.): EDBT 1988. LNCS, vol. 303. Springer, Heidelberg (1988)
[3] McLuhan, M.: Understanding Media: The Extensions of Man. The MIT Press, Cambridge (1994)
[4] Schmidt, J.W.: Some high-level language constructs for data of type relation. ACM Transactions on Database Systems, 247–261 (1977)
[5] Matthes, F., Schmidt, J.W.: Bulk Types: Built-In or Add-On? In: Kanelakis, P., Schmidt, J.W. (eds.) Proc. Third InternationalWorkshop on Database Programming Languages, Nafplion, Greece, Morgan Kaufmann Publishers, San Francisco (1991)
[6] Mathiske, B., Matthes, F., Schmidt, J.W.: On Migrating Threads. Journal of Intelligent Information Systems 8(2), 167–191 (1997)
[7] Brodie, M., Mylopoulos, J., Schmidt, J.W.: On conceptual modelling - perspectives from artificial intelligence, databases, and programming languages. Topics in information systems. Springer, Heidelberg (1984)

[8] Schmidt, J.W., Wetzel, I., Borgida, A., Mylopoulos, J.: Database programming by formal refinement of conceptual designs. IEEE – Data Engineering (1989)
[9] Warnke, M., Fleckner, U., Ziegler, H.: Handbuch zur Politischen Ikonographie. Verlag Beck, München (2009)
[10] Cassirer, E.: Zur Logik der Kulturwissenschaften, Göteborg (1942); see also: Wissenschaftliche Buchgesellschaft, Darmstadt (1961); The Logic of the Cultural Sciences, Yale University Press (2000)
[11] Schmitz-Rigal, C.: Die Kunst offenen Wissens (The Art of Open Knowledge). Felix Meiner Verlag, Hamburg (2002)
[12] Deacon, T.W.: The Symbolic Species. The Co-evolution of Language and the Brain. W.W. Norton Company, New York (1997)
[13] Wollheim, R.: Art and its Objects. Harper and Row, New York (1968); see also: Objekte der Kunst. Suhrkamp Verlag, Frankfurt am Main (1982)
[14] Gawecki, A., Matthes, F., Schmidt, J.W., Stamer, S.: CoreMedia AG. Co-founder of CoreMedia AG, Hamburg (1998); http://www.coremedia.com
[15] Sehring, H.-W., Bossung, S., Schmidt, J.W.: Active learning by personalization - lessons learnt from research in conceptual content management. In: Proc. of the 1st International Conference on Web Information Systems and Technologies, pp. 496–503. INSTICC Press, Miami (2005)
[16] Sehring, H.-W., Bossung, S., Schmidt, J.W.: Collaborative e-learning processes in conceptual content management systems. In: Proc. of the IADIS International Conference on Cognition and Exploratory Learning in Digital Age, IADIS, pp. 397–400 (2007)
[17] Hastie, T., Tibshirani, R., Friedman, J.: The elements of Statistical Learning: Data Mining, Inference and Prediction. Springer, Heidelberg (2001)
[18] Mylopoulos, J., Chung, L., Nixon, B.: Representing and using non-functional requirements: A process-oriented approach. IEEE Transactions on Software Engineering, Special Issue on Knowledge Representation and Reasoning in Software Development, 483–497 (1992)
[19] Chung, L., Nixon, B., Yu, E., Mylopoulos, J.: Non-functional Requirements in Software Engineering. Kluwer, Dordrecht (2000)
[20] Sehring, H.-W., Schmidt, J.W.: Beyond Databases: An Asset Language for Conceptual Content Management. In: Benczúr, A.A., Demetrovics, J., Gottlob, G. (eds.) ADBIS 2004. LNCS, vol. 3255, pp. 99–112. Springer, Heidelberg (2004)
[21] Wittgenstein, L.: Philosophische Untersuchungen. Suhrkamp Verlag, Frankfurt am Main (1960)
[22] Goodman, N.: Languages of Art, New York (1968); see also: Sprachen der Kunst. Suhrkamp-Verlag, Frankfurt am Main (1969)
[23] Révész, G.: Lambda-Calculus, Combinators, and Functional Programming. Cambridge University Press, Cambridge (1988)
[24] Bossung, S.: Conceptual Content Modeling: Languages, Applications, and Systems. PhD thesis, Hamburg University of Science and Technology (2007)
[25] Warburg, A.: Italienische Kunst und internazionale Astrologie im Palazzo Schifanoja zu Ferrara, Gesammelte Schriften, vol. 2. Leipzig-Berlin (1932); see also: Heckscher, W.S.: The Genesis of Iconology (1964)
[26] Panofsky, E.: Zum Problem der Beschreibung und Inhaltsdeutung von Werken der bildenden Kunst. Logos 21 (1932); see also: Aufsätze zu Grundfragen der Kunstwissenschaft, Berlin (1964)
[27] Panofsky, E.: Iconography and Iconology: An Introduction into the Study of Renaissance Art. Meaning in the Visual Arts. Doubleday Anchor Books, Garden City, NY (1955)

[28] Panofsky, E.: Meaning in the Visual Arts. Doubleday Anchor Books, Garden City, NY (1955)
[29] Svenonius, E.: The Intellectual Foundation of Information Organization. The MIT Press, Cambridge (2000)
[30] Schmidt, J.W., Schroeder, G., Niederee, C., Matthes, F.: Linguistic and architectural requirements for personalized digital libraries. International Journal on Digital Libraries, 89–104 (1997)
[31] Bossung, S., Sehring, H.-W., Hupe, P., Schmidt, J.W.: Open and dynamic schema evolution in content-intensive web applications. In: Proc. of the Second International Conference on Web Information Systems and Technologies, pp. 109–116. INSTICC Press (2006)
[32] Dittmann, L.: Zur Kritik der kunstwissenschaftlichen Symboltheorie. Ikonographie und Ikonologie. DuMont Verlag, Köln (1967)
[33] Forssmann, E.: Ikonologie und allgemeine Kunstgeschichte. Ikonographie und Ikonologie. DuMont Verlag, Köln (1967)
[34] Schneider, W.: Ausdruckswerte der Deutschen Sprache – Eine Stilkunde. Wissenschaftliche Buchgesellschaft, Darmstadt (1968), Expression Values of the German Language - Elements of Style, Leipzig und Berlin (1931)
[35] Sehring, H.-W.: Konzeptorientierte Inhaltsverwaltung – Modell, Systemarchitektur und Prototypen. Dissertation, Hamburg University of Science and Technology (2003)
[36] Sehring, H.-W., Bossung, S., Schmidt, J.W.: Die Warburg Electronic Library: Materialien und Dokumenten zur Geschichte der Kunstgeschichte im Nationalsozialismus. Schriften zur modernen Kunsthistoriographie, pp. 39–61 (2008)
[37] Sehring, H.-W., Bossung, S., Schmidt, J.W.: Content is capricious: A case for dynamic system generation. In: Manolopoulos, Y., Pokorný, J., Sellis, T.K. (eds.) ADBIS 2006. LNCS, vol. 4152, pp. 430–445. Springer, Heidelberg (2006)
[38] Wiederhold, G.: Mediators in the Architecture of Future Information Systems. IEEE Computer, 38–49 (1992)
[39] Mezini, M., Seiter, L., Lieberherr, K.: Component integration with pluggable composite adapters. In: Software Architectures and Component Technology, Kluwer, Dordrecht (2000)

Conceptual Modeling for Data Integration

Diego Calvanese[1], Giuseppe De Giacomo[2], Domenico Lembo[2], Maurizio Lenzerini[2], and Riccardo Rosati[2]

[1] KRDB Research Centre
Free University of Bozen-Bolzano
calvanese@inf.unibz.it
[2] Dip. di Informatica e Sistemistica
SAPIENZA Università di Roma
lastname@dis.uniroma1.it

Abstract. The goal of data integration is to provide a uniform access to a set of heterogeneous data sources, freeing the user from the knowledge about where the data are, how they are stored, and how they can be accessed. One of the outcomes of the research work carried out on data integration in the last years is a clear architecture, comprising a global schema, the source schema, and the mapping between the source and the global schema. Although in many research works and commercial tools the global schema is simply a data structure integrating the data at the sources, we argue that the global schema should represent, instead, the conceptual model of the domain. However, to fully pursue such an approach, several challenging issues are to be addressed. The main goal of this paper is to analyze one of them, namely, how to express the conceptual model representing the global schema. We start our analysis with the case where such a schema is expressed in terms of a UML class diagram, and we end up with a proposal of a particular Description Logic, called $\textit{DL-Lite}_{\mathcal{A},id}$. We show that the data integration framework based on such a logic has several interesting properties, including the fact that both reasoning at design time, and answering queries at run time can be done efficiently.

1 Introduction

The goal of data integration is to provide a uniform access to a set of heterogeneous data sources, freeing a client from the knowledge about where the data are, how they are stored, and how they can be accessed. The problem of designing effective data integration solutions has been addressed by several research and development projects in the last years. However, data integration is still one of the major challenges in Information Technology [5]. One of the reasons is that large amounts of heterogeneous data are nowadays available within an organization, but these data have been often collected and stored by different applications and systems. Therefore, on the one hand the need of accessing data by means of flexible and unified mechanisms is becoming more and more important, and, on the other hand, current commercial data integration tools have several drawbacks.

Starting from the late 90s, research in data integration has mostly focused on declarative approaches (as opposed to procedural ones) [32,26]. One of the outcomes of this research work is a clear architecture for (mediator-based[1]) data integration. According to this architecture [26], the main components of a data integration system are the global schema, the sources, and the mapping. Roughly speaking, the sources represent the repositories where the data are, the global schema represents the unified structure presented to the client, and the mapping relates the source data with the global schema. There are at least two different approaches to the design of the global schema. In the first approach, the global schema is expressed in terms of a database model (e.g., the relational model, or a semistructured data model), and represents a sort of unified data structure accommodating the various data at the sources. In the second approach the global schema provides a conceptual representation of the application domain [6], rather than a specification of a data structure. Thus, in this approach the distinction between the global schema and the data sources reflects the separation between the conceptual level (the one presented to the client), and the logical/physical level of the information system (the one stored in the sources), with the mapping acting as the reconciling structure between the two levels.

Although most of the research projects and the commercial data integration tools follow the first approach, we argue that designing the system in such a way that the global schema represents the conceptual model of the domain has several interesting advantages, both in the design and in the operation of the data integration system.

The first advantage is that a conceptual level in the architecture for data integration is the obvious means for pursuing a declarative approach to integration. As a consequence, all the advantages deriving from making various aspects of the system explicit are obtained, including the crucial fact that the conceptual level provides a system independent specification of the domain of interest to the client. By making the representation of the domain explicit we gain re-usability of the acquired knowledge, which is not achieved when the global schema is simply a unified description of the underlying data sources. This may also have consequences on the design of the user interface. Indeed, conceptual models are naturally expressed in a graphical form, and graphical tools that adequately present the overall information scenario are key factors in user interfaces.

A second advantage is the use of the mapping component of the system for explicitly specifying the relationships between the data sources and the domain concepts. The importance of this clearly emerges when looking at large organizations where the information about data is widespread into separate pieces of documentation that are often difficult to access and non necessarily conforming to common standards. The conceptual model built for data integration can thus provide a common ground for the documentation of the enterprise data stores and can be seen as a formal specification for mediator design. Obviously, the above advantages carry over in the maintenance phase of the data integration system (sources change, hence "design never ends").

A third advantage has to do with incrementality and extensibility of the system. One criticism that is often raised to mediator-based data integration is that it requires merging and integrating the source data, and this merging process can be very costly. However, the conceptual approach to data integration does not impose to fully integrate the data sources at once. Rather, after building the domain model, one can incrementally add new sources or new elements therein, when they become available, or when needed,

[1] Other architectures, e.g. [4], are outside the scope of this paper.

thus amortizing the cost of integration. Therefore, the overall design can be regarded as the incremental process of understanding and representing the domain on the one hand, and the available data on the other hand.

We believe that all the advantages outlined above represent convincing arguments supporting the conceptual approach to data integration. However, to fully pursue such an approach, several challenging issues are to be addressed. The goal of this paper is to analyze one of them, namely, how to express the conceptual model representing the global schema.

We start our analysis with the case where the global schema of the data integration system is expressed in terms of a UML class diagram. Notably, we show that the expressive power of UML class diagrams is enough to get intractability of various tasks, including query answering. We then present a specific proposal of a logic-based language for expressing conceptual models. The language, called *DL-Lite*$_{\mathcal{A},id}$, is a tractable Description Logic, specifically defined for achieving tractability of the reasoning tasks that are relevant in data integration. The proposed data integration framework, based on such a logic, has several interesting properties, including the fact that both reasoning at design time, and answering queries at run time can be done efficiently. Also, we study possible extensions to the data integration framework based on *DL-Lite*$_{\mathcal{A},id}$, and show that our proposal basically represents an optimal compromise between expressive power and computational complexity.

The paper is organized as follows. In Section 2, we describe a general architecture for data integration, and the basic features of Description Logics, which are the logics we use to formally express conceptual models. In Section 3, we analyze the case where the global schema of the data integration system is expressed in terms of a UML class diagram. In Section 4, we illustrate the basic characteristics of the Description Logic *DL-Lite*$_{\mathcal{A},id}$, and in Section 5, we illustrate a specific proposal of data integration system based on such a logic. In Section 6, we study possible extensions to such data integration framework, whereas in Section 7, we conclude the paper with a discussion on related and future work.

2 The Data Integration Framework

The goal of this section is to describe a general architecture for data integration. We restrict our attention to data integration systems based on a so-called global schema, or mediated schema. In other words, we refer to data integration systems whose aim is providing the user with a representation of the domain of interest to the system, and connecting such a representation to the data residing at the sources. Thus, the global schema implicitly provides a reconciled view of all data, which can be queried by the user. One of the theses of this paper is that the global schema can be profitably expressed in terms of a conceptual model of the domain, and such a conceptual model can be formalized in specialized logics, called Description Logics. Therefore, another goal of this section is to illustrate the basic features of such logics.

2.1 Architecture for Data Integration

According to [26], we formalize a *data integration system* $\mathcal{J}$ in terms of a triple $\langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$, where

- $\mathcal{G}$ is the *global schema*, expressed in a language $\mathcal{L}_\mathcal{G}$ over an alphabet $\mathcal{A}_\mathcal{G}$. The alphabet comprises a symbol for each element of $\mathcal{G}$ (i.e., relation if $\mathcal{G}$ is relational, class if $\mathcal{G}$ is object-oriented, etc.).
- $\mathcal{S}$ is the *source schema*, expressed in a language $\mathcal{L}_\mathcal{S}$ over an alphabet $\mathcal{A}_\mathcal{S}$. The alphabet $\mathcal{A}_\mathcal{S}$ includes a symbol for each element of the sources.
- $\mathcal{M}$ is the *mapping* between $\mathcal{G}$ and $\mathcal{S}$, constituted by a set of *assertions* of the forms

$$q_\mathcal{S} \leadsto q_\mathcal{G},$$
$$q_\mathcal{G} \leadsto q_\mathcal{S},$$

  where $q_\mathcal{S}$ and $q_\mathcal{G}$ are two queries of the same arity, respectively over the source schema $\mathcal{S}$, and over the global schema $\mathcal{G}$. Queries $q_\mathcal{S}$ are expressed in a query language $\mathcal{L}_{\mathcal{M},\mathcal{S}}$ over the alphabet $\mathcal{A}_\mathcal{S}$, and queries $q_\mathcal{G}$ are expressed in a query language $\mathcal{L}_{\mathcal{M},\mathcal{G}}$ over the alphabet $\mathcal{A}_\mathcal{G}$. Intuitively, an assertion $q_\mathcal{S} \leadsto q_\mathcal{G}$ specifies that the concept represented by the query $q_\mathcal{S}$ over the sources corresponds to the concept in the global schema represented by the query $q_\mathcal{G}$ (similarly for an assertion of type $q_\mathcal{G} \leadsto q_\mathcal{S}$).

The global schema provides a description of the domain of interest, and not simply a unified representation of the source data. The source schema describes the structure of the sources, where the real data are. The assertions in the mapping establish the connection between the elements of the global schema and those of the source schema.

The semantics of a data integration system is based on the notion of interpretation in logic. Indeed, in this paper we assume that $\mathcal{G}$ is formalized as a logical theory, and therefore, given a source database D (i.e., a database for $\mathcal{S}$), the semantics of the whole system coincides with the set of interpretations that satisfy all the assertions of $\mathcal{G}$ (i.e., they are logical models of $\mathcal{G}$) and all the assertions of $\mathcal{M}$ with respect to D. Such a set of interpretations, denoted $sem_D(\mathcal{J})$, is called the set of models of $\mathcal{J}$ relative to D.

There are two basic tasks concerning a data integration system that we consider in this paper. The first task is relevant in the design phase of the system, and concerns the possibility of reasoning over the global schema: given $\mathcal{G}$ and a logical assertion α, check whether α holds in every model of $\mathcal{G}$. The second task is query answering, which is crucial during the run-time of the system. Queries to $\mathcal{J}$ are posed in terms of the global schema $\mathcal{G}$, and are expressed in a query language $\mathcal{L}_Q$ over the alphabet $\mathcal{A}_\mathcal{G}$. A query is intended to provide the specification of which extensional information to extract from the domain of interest in the data integration system. More precisely, given a source database D, the answer $q^{\mathcal{J},D}$ to a query q in $\mathcal{J}$ with respect to D is the set of tuples t of objects such that $t \in q^\mathcal{B}$ (i.e., t is an answer to q over $\mathcal{B}$) for *every* model $\mathcal{B}$ of $\mathcal{J}$ relative to D. The set $q^{\mathcal{J},D}$ is called the set of *certain answers* to q in $\mathcal{J}$ with respect to D. Note that, from the point of view of logic, finding certain answers is a logical implication problem: check whether the fact that t satisfies the query logically follows from the information on the sources and on the mapping.

The above definition of data integration system is general enough to capture virtually all approaches in the literature. Obviously, the nature of a specific approach depends on the characteristics of the mapping, and on the expressive power of the various schema and query languages. For example, the language $\mathcal{L}_\mathcal{G}$ may be very simple (basically allowing for the definition of a set of relations), or may allow for various forms of integrity constraints to be expressed over the symbols of $\mathcal{A}_{\mathcal{G}}$. Analogously, the type (e.g., relational, semistructured, etc.) and the expressive power of $\mathcal{L}_{\mathcal{S}}$ varies from one approach to another.

2.2 Description Logics

Description Logics [2] (DLs) were introduced in the early 80s in the attempt to provide a formal ground to Semantic Networks and Frames. Since then, they have evolved into knowledge representation languages that are able to capture virtually all class-based representation formalisms used in Artificial Intelligence, Software Engineering, and Databases. One of the distinguishing features of the work on these logics is the detailed computational complexity analysis both of the associated reasoning algorithms, and of the logical implication problem that the algorithms are supposed to solve. By virtue of this analysis, most of these logics have optimal reasoning algorithms, and practical systems implementing such algorithms are now used in several projects. In DLs, the domain of interest is modeled by means of *concepts* and *roles* (i.e., binary relationships), which denote classes of objects and binary relations between classes of objects, respectively. Concepts and roles can be denoted using expressions of a specified languages, and the various DLs differ in the expressive power of such a language. The DLs considered in this paper are subsets of a DL called $\mathcal{ALCQI}b_{id}$. $\mathcal{ALCQI}b_{id}$ is an expressive DL that extends the basic DL language $\mathcal{AL}$ (attributive language) with negation of arbitrary concepts (indicated by the letter $\mathcal{C}$), qualified number restrictions (indicated by the letter $\mathcal{Q}$), inverse of roles (indicated by the letter $\mathcal{I}$), boolean combinations of roles (indicated by the letter b), and identification assertions (indicated by the subscript *id*). More in detail, concepts and roles in $\mathcal{ALCQI}b_{id}$ are formed according to the following syntax:

$$\begin{array}{rl} C, C' \longrightarrow & A \mid \neg C \mid C \sqcap C' \mid C \sqcup C' \mid \\ & \forall R.C \mid \exists R.C \mid \geqslant n\, R.C \mid \leqslant n\, R.C \\ R, R' \longrightarrow & P \mid P^- \mid R \cap R' \mid R \cup R' \mid R \setminus R' \end{array}$$

where A denotes an *atomic concept*, P an *atomic role*, P^- the *inverse* of an atomic role, C, C' arbitrary *concepts*, and R, R' arbitrary *roles*. Furthermore, $\neg C$ denotes concept negation, $C \sqcap C'$ concept intersection, $C \sqcup C'$ concept union, $\forall R.C$ value restriction, $\exists R.C$ qualified existential quantification on roles, and $\geqslant n\, R.C$ and $\leqslant n\, R.C$ so-called number restrictions. We use $\top$, denoting the top concept, as an abbreviation for $A \sqcup \neg A$, for some concept A. An arbitrary role can be an atomic role or its inverse, or a role obtained combining roles through set theoretic operators, i.e., intersection ("$\cap$"), union ("$\cup$"), and difference ("$\setminus$"). W.l.o.g., we assume difference applied only to atomic roles and their inverses.

As an example, consider the atomic concepts *Man* and *Woman*, and the atomic roles *HAS-HUSBAND*, representing the relationship between a woman and the man with whom she is married, and *HAS-CHILD*, representing the parent-child relationship. Then, intuitively, the inverse of *HAS-HUSBAND*, i.e., *HAS-HUSBAND*$^-$, represents the relationship between a man and his wife. Also, *Man* $\sqcup$ *Woman* is a concept representing people (considered the union of men and women), whereas the concept $\exists$*HAS-CHILD*.*Woman* represents those having a daughter, and the concept

$\geqslant 2$ *HAS-CHILD*.*Woman* $\sqcap$ $\leqslant 4$ *HAS-CHILD*.$\top$ represents those having at least two daughters and at most four children.

Like in any DL, an $\mathcal{ALCQIb}_{id}$ *knowledge base* (KB) is a pair $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$, where $\mathcal{T}$, the *TBox*, is a finite set of *intensional assertions*, and $\mathcal{A}$, the *ABox*, is a finite set of *extensional* (or, *membership*) *assertions*.

The TBox may contain intensional assertions of two types, namely inclusion assertions, and local identification assertions (see [13] for the meaning of "local" in this context).

– An *inclusion assertion* has the form $C \sqsubseteq C'$, with C and C' arbitrary $\mathcal{ALCQIb}_{id}$ concepts, or the form $R \sqsubseteq R'$, with R and R' arbitrary $\mathcal{ALCQIb}_{id}$ roles. Intuitively, an inclusion assertion states that, in every model of $\mathcal{T}$, each instance of the left-hand side expression is also an instance of the right-hand side expression. For example, the inclusions *Woman* $\sqsubseteq$ $\leqslant 1$ *HAS-HUSBAND*.$\top$ and $\exists$*HAS-HUSBAND*$^{-}$.$\top$ $\sqsubseteq$ Man respectively specifies that women may have at most one husband and that husbands are men.
– A *local identification assertion* (or, simply, identification assertion or identification constraint – IdC) makes use of the notion of path. A *path* is an expression built according to the following syntax,

$$\pi \longrightarrow S \mid D? \mid \pi \circ \pi \qquad (1)$$

where S denotes an atomic role or the inverse of an atomic role, and $\pi_1 \circ \pi_2$ denotes the composition of the paths π_1 and π_2. Finally, D denotes a concept, and the expression $D?$ is called a *test relation*, which represents the identity relation on instances of D. Test relations are used in all those cases in which one wants to impose that a path involves instances of a certain concept. For example, *HAS-CHILD* $\circ$ *Woman*? is the path connecting someone with his/her daughters.

A path π denotes a complex property for the instances of concepts: given an object o, every object that is reachable from o by means of π is called a π-*filler* for o. Note that for a certain o there may be several distinct π-fillers, or no π-fillers at all.

If π is a path, the length of π, denoted $\mathit{length}(\pi)$, is 0 if π has the form $D?$, is 1 if π has the form S, and is $\mathit{length}(\pi_1) + \mathit{length}(\pi_2)$ if π has the form $\pi_1 \circ \pi_2$. With the notion of path in place, we are ready for the definition of local identification assertion, which is an assertion of the form

$$(\mathsf{id}\ C\ \pi_1, \ldots, \pi_n)$$

where C is an arbitrary concept, $n \geq 1$, and $\pi_1, \ldots, \pi_n$ (called the *components* of the identifier) are paths such that $\mathit{length}(\pi_i) \geq 1$ for all $i \in \{1, \ldots, n\}$, and $\mathit{length}(\pi_i) = 1$ for at least one $i \in \{1, \ldots, n\}$. Intuitively, such a constraint asserts that for any two different instances o, o' of C, there is at least one π_i such that o and o' differ in the set of their π_i-fillers. The term "local" emphasizes that at least one of the paths refers to a local property of C.

For example, the identification assertion $(\mathsf{id}$ *Woman HAS-HUSBAND*$)$ says that a woman is identified by her husband, i.e., there are not two different women with

$$\begin{aligned}
A^{\mathcal{I}} &\subseteq \Delta^{\mathcal{I}} \\
\neg C^{\mathcal{I}} &= \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}} \\
(C \sqcap C')^{\mathcal{I}} &= C^{\mathcal{I}} \cap C'^{\mathcal{I}} \\
(C \sqcup C')^{\mathcal{I}} &= C^{\mathcal{I}} \cap C'^{\mathcal{I}} \\
(\forall R.C)^{\mathcal{I}} &= \{\, o \mid \forall o'.\, (o,o') \in R^{\mathcal{I}} \supset o' \in C^{\mathcal{I}} \,\} \\
(\exists R.C)^{\mathcal{I}} &= \{\, o \mid \exists o'.\, (o,o') \in R^{\mathcal{I}} \wedge o' \in C^{\mathcal{I}} \,\} \\
(\geqslant n\, R.C)^{\mathcal{I}} &= \{\, o \mid |\{o' \in C^{\mathcal{I}} \mid (o,o') \in R^{\mathcal{I}}\}| \geq n \,\} \\
(\leqslant n\, R.C)^{\mathcal{I}} &= \{\, o \mid |\{o' \in C^{\mathcal{I}} \mid (o,o') \in R^{\mathcal{I}}\}| \leq n \,\}
\end{aligned}
\qquad
\begin{aligned}
P^{\mathcal{I}} &\subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}} \\
(P^-)^{\mathcal{I}} &= \{(o,o') \mid (o',o) \in P^{\mathcal{I}}\} \\
(R \cap R')^{\mathcal{I}} &= R^{\mathcal{I}} \cap R'^{\mathcal{I}} \\
(R \cup R')^{\mathcal{I}} &= R^{\mathcal{I}} \cup R'^{\mathcal{I}} \\
(R \setminus R')^{\mathcal{I}} &= R^{\mathcal{I}} \setminus R'^{\mathcal{I}}
\end{aligned}$$

Fig. 1. Interpretation of $\mathcal{ALCQI}b_{id}$ concepts and roles

the same husband, whereas the identification assertion $(\mathsf{id}$ *Man HAS-CHILD*$)$ says that a man is identified by his children, i.e., there are not two men with a child in common. We can also say that there are not two men with the same daughters by means of the identification $(\mathsf{id}$ *Man HAS-CHILD* $\circ$ *Woman*$?)$.

The ABox consists of a set of extensional assertions, which are used to state the instances of concepts and roles. Each such assertion has the form $A(a)$, $P(a,b)$, $a = b$, or $a \neq b$, with A and P respectively an atomic concept and an atomic role occurring in $\mathcal{T}$, and a, b constants.

We now turn to the semantics of $\mathcal{ALCQI}b_{id}$, which is given in terms of interpretations. An *interpretation* $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ consists of a non-empty interpretation domain $\Delta^{\mathcal{I}}$ and an interpretation function $\cdot^{\mathcal{I}}$, which assigns to each concept C a subset $C^{\mathcal{I}}$ of $\Delta^{\mathcal{I}}$, and to each role R a binary relation $R^{\mathcal{I}}$ over $\Delta^{\mathcal{I}}$ is such a way that the conditions specified in Figure 1 are satisfied. The semantics of an $\mathcal{ALCQI}b_{id}$ KB $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ is the set of *models* of $\mathcal{K}$, i.e., the set of interpretations satisfying all assertions in $\mathcal{T}$ and $\mathcal{A}$. It remains to specify when an interpretation satisfies an assertion.

An interpretation $\mathcal{I}$ *satisfies* an inclusion assertion $C \sqsubseteq C'$ (resp., $R \sqsubseteq R'$), if $C^{\mathcal{I}} \subseteq C'^{\mathcal{I}}$ (resp., $R^{\mathcal{I}} \subseteq R'^{\mathcal{I}}$).

In order to define the semantics of IdCs, we first define the semantics of paths, and then specify the conditions for an interpretation to satisfy an IdC. The extension $\pi^{\mathcal{I}}$ of a path π in an interpretation $\mathcal{I}$ is defined as follows:

– if $\pi = S$, then $\pi^{\mathcal{I}} = S^{\mathcal{I}}$,
– if $\pi = D?$, then $\pi^{\mathcal{I}} = \{\, (o,o) \mid o \in D^{\mathcal{I}} \,\}$,
– if $\pi = \pi_1 \circ \pi_2$, then $\pi^{\mathcal{I}} = \pi_1^{\mathcal{I}} \circ \pi_2^{\mathcal{I}}$, where $\circ$ denotes the composition operator on relations.

As a notation, we write $\pi^{\mathcal{I}}(o)$ to denote the set of π-fillers for o in $\mathcal{I}$, i.e., $\pi^{\mathcal{I}}(o) = \{o' \mid (o,o') \in \pi^{\mathcal{I}}\}$. Then, an interpretation $\mathcal{I}$ satisfies the IdC $(\mathsf{id}\ C\ \pi_1, \ldots, \pi_n)$ if for all $o, o' \in C^{\mathcal{I}}$, $\pi_1^{\mathcal{I}}(o) \cap \pi_1^{\mathcal{I}}(o') \neq \emptyset \wedge \cdots \wedge \pi_n^{\mathcal{I}}(o) \cap \pi_n^{\mathcal{I}}(o') \neq \emptyset$ implies $o = o'$. Observe that this definition is coherent with the intuitive reading of IdCs discussed above, in particular by sanctioning that two different instances o, o' of C differ in the set of their π_i-fillers when such sets are disjoint.

Finally, to specify the semantics of $\mathcal{ALCQI}b_{id}$ ABox assertions, we extend the interpretation function to constants, by assigning to each constant a an object $a^{\mathcal{I}} \in \Delta^{\mathcal{I}}$.

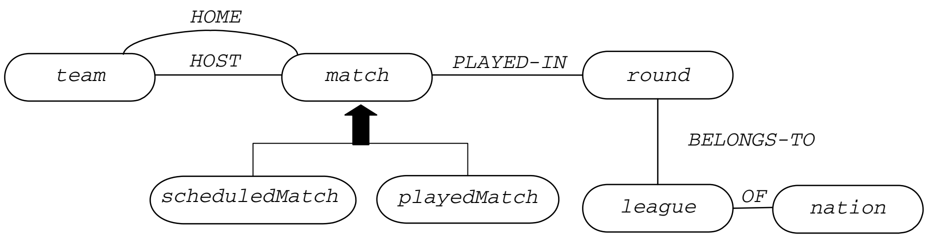


Fig. 2. Diagrammatic representation of the football leagues example

Inclusion assertions	
$\mathit{league} \sqsubseteq \exists \mathit{OF}.\top$	$\exists \mathit{HOST}.\top \sqsubseteq \mathit{match}$
$\exists \mathit{OF}.\top \sqsubseteq \mathit{league}$	$\exists \mathit{HOST}^-.\top \sqsubseteq \mathit{team}$
$\exists \mathit{OF}^-.\top \sqsubseteq \mathit{nation}$	$\mathit{match} \sqsubseteq \exists \mathit{HOST}.\top$
$\mathit{round} \sqsubseteq \exists \mathit{BELONGS\text{-}TO}.\top$	$\mathit{playedMatch} \sqsubseteq \mathit{match}$
$\exists \mathit{BELONGS\text{-}TO}.\top \sqsubseteq \mathit{round}$	$\mathit{scheduledMatch} \sqsubseteq \mathit{match}$
$\exists \mathit{BELONGS\text{-}TO}^-.\top \sqsubseteq \mathit{league}$	$\mathit{playedMatch} \sqsubseteq \neg \mathit{scheduledMatch}$
$\mathit{match} \sqsubseteq \exists \mathit{PLAYED\text{-}IN}.\top$	$\mathit{match} \sqsubseteq \mathit{playedMatch} \sqcup \mathit{scheduledMatch}$
$\exists \mathit{PLAYED\text{-}IN}.\top \sqsubseteq \mathit{match}$	$\top \sqsubseteq \leqslant 1\, \mathit{OF}.\top$
$\exists \mathit{PLAYED\text{-}IN}^-.\top \sqsubseteq \mathit{round}$	$\top \sqsubseteq \leqslant 1\, \mathit{BELONGS\text{-}TO}.\top$
$\mathit{match} \sqsubseteq \exists \mathit{HOME}.\top$	$\top \sqsubseteq \leqslant 1\, \mathit{PLAYED\text{-}IN}.\top$
$\exists \mathit{HOME}.\top \sqsubseteq \mathit{match}$	$\top \sqsubseteq \leqslant 1\, \mathit{HOME}.\top$
$\exists \mathit{HOME}^-.\top \sqsubseteq \mathit{team}$	$\top \sqsubseteq \leqslant 1\, \mathit{HOST}.\top$
Identification assertions	
(id *match* *HOME*, *PLAYED-IN*)	
(id *match* *HOST*, *PLAYED-IN*)	

Fig. 3. The TBox in $\mathcal{ALCQI}b_{id}$ for the football leagues example

An interpretation $\mathcal{I}$ satisfies a membership assertion $A(a)$ if $a^\mathcal{I} \in A^\mathcal{I}$, a membership assertion $P(a, b)$ if $(a^\mathcal{I}, b^\mathcal{I}) \in P^\mathcal{I}$, an assertion of the form $a = b$ if $a^\mathcal{I} = b^\mathcal{I}$, and an assertion of the form $a \neq b$ if $a^\mathcal{I} \neq b^\mathcal{I}$.

We also note that, as in many DLs, reasoning in $\mathcal{ALCQI}b_{id}$, i.e., checking whether an assertion holds in every model of a KB, is decidable in deterministic exponential time (see [2]).

We conclude this section with an example in which we present an $\mathcal{ALCQI}b_{id}$ TBox modeling the annual national football[2] championships in Europe, where the championship for a specific nation is called *league* (e.g., the Spanish Liga). A league is structured in terms of a set of *rounds*. Every round contains a set of *matches*, each one characterized by one *home team* and one *host team*. We distinguish between scheduled matches, i.e., matches that have still to be played, and played matches. Obviously, a match falls in exactly one of these two categories.

In Figure 2, we show a schematic representation of (part of) the ontology for the football leagues domain. In this figure, the black arrow represents a partition of one concept into a set of sub-concepts. The TBox assertions in $\mathcal{ALCQI}b_{id}$ capturing the above aspects are shown in Figure 3. In particular, the identification constraints model the fact that a team is the home team of at most one match per round, and the host team of at most one match per round.

[2] Football is called “soccer” in the United States.

3 UML Class Diagram as Global Schema

In this section, we discuss the case where the global schema of a data integration system is a UML class diagram.

Since we concentrate on class diagrams from the data integration perspective, we do not deal with those features that are more relevant for the software engineering perspective, such as operations (methods) associated to classes, or public, protected, and private qualifiers for methods and attributes. Also, for sake of brevity and to smooth the presentation we make some simplifying assumptions that could all be lifted without changing the results presented here (we refer to [3] for further details). In particular, we will not deal explicitly with associations of arity greater than 2, and we will only deal with the following multiplicities: $0..*$ (unconstrained), functional participation $0..1$, mandatory participation $1..*$, and one-to-one correspondence $1..1$. These multiplicities are particularly important since they convey meaningful semantic aspects in modeling, and thus are the most commonly used ones.

Our goal is twofold. One the one hand, we aim at showing how class diagrams can be expressed in DLs. On the other hand, we aim at understanding which is the complexity of the two tasks we are interested in for a data integration system, when the global schema is a UML class diagram. We will show that the formalization in DLs helps us in deriving complexity results for both tasks.

3.1 Representing UML Class Diagrams in DLs

A *class* in a UML class diagram denotes a *sets of objects* with common features. The specification of a class contains its *name* and its *attributes*, each denoted by a name (possibly followed by the *multiplicity*, between square brackets) and with an associated *type*, which indicates the domain of the attribute values.

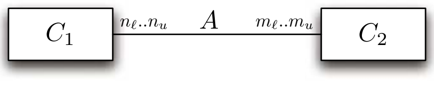
A UML class is represented by a DL concept. This follows naturally from the fact that both UML classes and DL concepts denote *sets of objects*.

A UML *attribute* a of type T for a class C associates to each instance of C, zero, one, or more instances of a class T. An optional *multiplicity* $[i..j]$ for a specifies that a associates to each instance of C, at least i and most j instances of T. When the multiplicity for an attribute is missing, $[1..1]$ is assumed, i.e., the attribute is *mandatory* and *single-valued*.

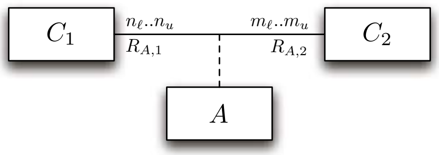
To formalize attributes, we have to think of an attribute a of type T for a class C as a binary relation between instances of C and instances of T. We capture such a binary relation by means of a DL role a_c. To specify the type of the attribute we use the DL assertions

$$\exists a_C \sqsubseteq C, \qquad\qquad \exists a_C^- \sqsubseteq T.$$

Such assertions specify precisely that, for each instance (c, v) of the role a_C, the object c is an instance of C, and the value v is an instance of T. Notice that in DL, the type T is represented as a concept, although containing values. Note that the attribute name a is not necessarily unique in the whole diagram, and hence two different classes, say C and C' could both have attribute a, possibly of different types. This situation is correctly captured in the DL formalization, where the attribute is contextualized to each class with a distinguished role, i.e., a_C and $a_{C'}$.



(a) Without association class



(b) With association class

Fig. 4. Association in UML

To specify that the attribute is mandatory (i.e., multiplicity $[1..*]$), we add the assertion

$$C \sqsubseteq \exists a_C,$$

which specifies that each instance of C participates necessarily at least once to the role a_C. To specify that the attribute is single-valued (i.e., multiplicity $[0..1]$), we add the assertion

$$(\mathsf{funct}\ a_C),$$

which is an abbreviation for $\top \sqsubseteq \leqslant 1\, a_C.\top$. Finally, if the attribute is both mandatory and single-valued (i.e., multiplicity $[1..1]$), we use both assertions together:

$$C \sqsubseteq \exists a_C, \qquad\qquad (\mathsf{funct}\ a_C).$$

An *association* in UML is a relation between the instances of two (or more) classes. An association often has a related *association class* that describes properties of the association, such as attributes, operations, etc. A binary association A between the instances of two classes C_1 and C_2 is graphically rendered as in Figure 4(a), where the *multiplicity* $m_\ell..m_u$ specifies that each instance of class C_1 can participate at least m_ℓ times and at most m_u times to association A. The multiplicity $n_\ell..n_u$ has an analogous meaning for class C_2.

An association A between classes C_1 and C_2 is formalized in DL by means of a role A on which we enforce the assertions

$$\exists A \sqsubseteq C_1, \qquad\qquad \exists A^- \sqsubseteq C_2.$$

To express the multiplicity $m_\ell..m_u$ on the participation of instances of C_2 for each given instance of C_1, we use the assertion $C_1 \sqsubseteq \exists A$, if $m_\ell = 1$, and $(\mathsf{funct}\ A)$, if $m_u = 1$. We can use similar assertions for the multiplicity $n_\ell..n_u$ on the participation of instances of C_1 for each given instance of C_2, i.e., $C_2 \sqsubseteq \exists A^-$, if $n_\ell = 1$, and $(\mathsf{funct}\ A^-)$, if $n_u = 1$.

Next we focus on an *association* with a related *association class*, as shown in Figure 4(b), where the class A is the association class related to the association, and $R_{A,1}$ and $R_{A,2}$, if present, are the *role names* of C_1 and C_2 respectively, i.e., they specify the role that each class plays within the association A.

We formalize in DL an association A with an association class, by reifying it into a DL concept A and introducing two DL roles $R_{A,1}$, $R_{A,2}$, one for each role of A, which intuitively connect an object representing an instance of the association to the instances of C_1 and C_2, respectively, that participate to the association[3]. Then, we enforce that each instance of A participates exactly once both to $R_{A,1}$ and to $R_{A,2}$, by means of the assertions

$$A \sqsubseteq \exists R_{A,1}, \qquad (\mathsf{funct}\ R_{A,1}), \qquad A \sqsubseteq \exists R_{A,2}, \qquad (\mathsf{funct}\ R_{A,2}).$$

To represent that the association A is between classes C_1 and C_2, we use the assertions

$$\exists R_{A,1} \sqsubseteq A, \qquad \exists R^-_{A,1} \sqsubseteq C_1, \qquad \exists R_{A,2} \sqsubseteq A, \qquad \exists R^-_{A,2} \sqsubseteq C_2.$$

Finally, we use the assertion

$$(\mathsf{id}\ A\ R_{A,1}, R_{A,2})$$

to specify that each instance of the concept A represents a *distinct* tuple in $C_1 \times C_2$.[4]

We can easily represent in DL multiplicities on an association with association class, by imposing suitable assertions on the inverses of the DL roles modeling the roles of the association. For example, to say that there is a one-to-one participation of instances of C_1 in the association (with related association class) A, we assert

$$C_1 \sqsubseteq \exists R^-_{A,1}, \qquad (\mathsf{funct}\ R^-_{A,1}).$$

In UML, one can use *generalization* between a parent class and a child class to specify that each instance of the child class is also an instance of the parent class. Hence, the instances of the child class inherit the properties of the parent class, but typically they satisfy additional properties that in general do not hold for the parent class.

Generalization is naturally supported in DLs. If a UML class C_2 generalizes a class C_1, we can express this by the DL assertion

$$C_1 \sqsubseteq C_2.$$

Inheritance between DL concepts works exactly as inheritance between UML classes. This is an obvious consequence of the semantics of $\sqsubseteq$, which is based on subsetting. As a consequence, in the formalization, each attribute of C_2 and each association involving C_2 is correctly inherited by C_1. Observe that the formalization in DL also captures directly inheritance among association classes, which are treated exactly as all other classes, and multiple inheritance between classes (including association classes).

Moreover in UML, one can group several generalizations into a class hierarchy, as shown in Figure 5. Such a hierarchy is captured in DL by a set of inclusion assertions, one between each child class and the parent class, i.e.,

$$C_i \sqsubseteq C, \qquad \text{for each } i \in \{1, \ldots, n\}.$$

Often, when defining generalizations between classes, we need to add additional assertions among the involved classes. For example, for the class hierarchy in Figure 5, an

[3] If the roles of the association are not available, we may use an arbitrary DL role name.

[4] Notice that such an approach can immediately be used to represent an association of any arity: it suffices to repeat the above for every component.

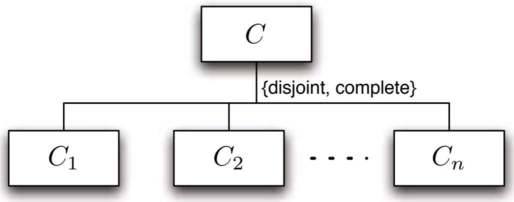


Fig. 5. A class hierarchy in UML

assertion may express that $C_1, \ldots, C_n$ are *mutually disjoint*. In DL, such a relationship can be expressed by the assertions

$$C_i \sqsubseteq \neg C_j, \qquad \text{for each } i, j \in \{1, \ldots, n\} \text{ with } i \neq j.$$

Moreover, we may want to express that a generalization hierarchy is *complete*, i.e., that the subclasses $C_1, \ldots, C_n$ are a *covering* of the superclass C. We can represent such a situation in DL by including the additional assertion

$$C \sqsubseteq C_1 \sqcup \cdots \sqcup C_n.$$

Such an assertion models a form of *disjunctive information*: each instance of C is either an instance of C_1, or an instance of C_2, $\ldots$ or an instance of C_n.

Similarly to generalization between classes, UML allows one to state *subset assertions* between associations. A subset assertion between two associations A and A' can be modeled in DL by means of the role inclusion assertion $A \sqsubseteq A'$, involving the two roles A and A' representing the associations. When the two associations A and A' are represented by means of association classes, we need to use the concept inclusion assertion $A \sqsubseteq A'$, together with the role inclusion assertions between corresponding roles of A and A'.

3.2 Reasoning and Query Answering

The fact that UML class diagrams can be captured by DLs enables the possibility of performing sound and complete reasoning to do formal verification at design time and query answering at runtime. Hence, one can exploit such ability to get support during the design phase of the global schema, and to take the information in the global schema fully into account during query answering.

It was shown in [3] that, unfortunately, reasoning (in particular checking the consistency of the diagram, a task to which other typical reasoning tasks of interest reduce) is EXPTIME-hard. What this result tells us is that, if the global schema is expressed in UML, then the support at design time for a data integration system may be impossible if the schema has a reasonable size.

Turning to query answering, the situation is even worse. The results in [11] imply that answering conjunctive queries in the presence of a UML class diagram formed by a single generalization with covering assertion is coNP-hard in the size of the instances of classes and associations. Hence, query answering over even moderately large data sets is again infeasible in practice. It is not difficult to see that this implies that, in a data integration system where the global schema is expressed as a UML diagram, answering conjunctive queries is coNP-hard with respect to the size of the source data.

Actually, as we will see in the next section, the culprit of such a high complexity is mainly the ability of expressing covering assertions, which induces reasoning by cases. Once we disallow covering and suitably restrict the simultaneous use of subset constraints between associations and multiplicities, not only the sources of exponential complexity disappear, but actually query answering becomes reducible to standard SQL evaluation over a relational database.

4 A Tractable DL: *DL-Lite*$_{\mathcal{A},id}$

We have seen that in a data integration system where the global schema is expressed as a UML class diagram, reasoning is too complex. Thus, a natural question arising at this point is: which is the right language to express the global schema of a data integration system?

In this section, we present *DL-Lite*$_{\mathcal{A},id}$, a DL of the *DL-Lite* family [12,11], enriched with identification constraints (idCs) [12], and show that it is very well suited for conceptual modeling in data integration, in particular for its ability of balancing expressive power with efficiency of reasoning, i.e., query answering, which can be managed through relational database technology.

DL-Lite$_{\mathcal{A},id}$ is essentially a subset of $\mathcal{ALCQI}b_{id}$, but, contrary to the DL presented in Section 2, it distinguishes concepts from *value-domains*, which denote sets of (data) values, and roles from *attributes*, which denote binary relations between objects and values. Concepts, roles, attributes, and value-domains in this DL are formed according to the following syntax[5]:

$$\begin{array}{lcl@{\qquad\qquad}lcl}
B & \longrightarrow & A \mid \exists Q \mid \delta(U) & E & \longrightarrow & \rho(U) \\
C & \longrightarrow & B \mid \neg B & F & \longrightarrow & \top_D \mid T_1 \mid \cdots \mid T_n \\
Q & \longrightarrow & P \mid P^- & V & \longrightarrow & U \mid \neg U \\
R & \longrightarrow & Q \mid \neg Q & & &
\end{array}$$

In such rules, A, P, and P^- respectively denote an *atomic concept*, an *atomic role*, and the *inverse of an atomic role*, Q and R respectively denote a *basic role* and an *arbitrary role*, whereas B denotes a *basic concept*, C an *arbitrary concept*, U an *atomic attribute*, V an *arbitrary attribute*, E a *basic value-domain*, and F an *arbitrary value-domain*. Furthermore, $\delta(U)$ denotes the *domain* of U, i.e., the set of objects that U relates to values; $\rho(U)$ denotes the *range* of U, i.e., the set of values that U relates to objects; $\top_D$ is the universal value-domain; $T_1, \ldots, T_n$ are n pairwise disjoint unbounded value-domains, corresponding to RDF data types, such as `xsd:string`, `xsd:integer`, etc.

[5] The results mentioned in this paper apply also to *DL-Lite*$_{\mathcal{A},id}$ extended with role attributes (cf. [9]), which are not considered here for the sake of simplicity.

In a *DL-Lite*$_{\mathcal{A},id}$ TBox, assertions have the forms

$$\begin{array}{cccc} B \sqsubseteq C & Q \sqsubseteq R & E \sqsubseteq F & U \sqsubseteq V \\ (\mathsf{funct}\ Q) & (\mathsf{funct}\ U) & (\mathsf{id}\ C\ \pi_1, \ldots, \pi_n) & \end{array}$$

The assertions above, from left to right, respectively denote inclusions between concepts, roles, value-domains, and attributes, (global) functionality on roles and on attributes, and identification constraints[6]. Notice that paths occurring in *DL-Lite*$_{\mathcal{A},id}$ identification assertions may involve also attributes and value-domains, which are instead not among the constructs present in $\mathcal{ALCQI}b_{id}$. More precisely, the symbol S in equation (1) now can be also an attribute or the inverse of an attribute, and the symbol D in (1) now can be also a basic or an arbitrary value-domain.

As for the ABox, beside assertions of the form $A(a)$, $P(a,b)$, with A an atomic concept, P and atomic role, and a, b constants, in *DL-Lite*$_{\mathcal{A},id}$ we may also have assertions of the form $U(a,v)$, where U is an atomic attribute, a a constant, and v a value. Notice however that assertions of the form $a = b$ or $a \neq b$ are not allowed.

We are now ready to define what a *DL-Lite*$_{\mathcal{A},id}$ KB is.

Definition 1. *A DL-Lite*$_{\mathcal{A},id}$ *KB* $\mathcal{K}$ *is a pair* $\langle \mathcal{T}, \mathcal{A} \rangle$*, where* $\mathcal{T}$ *is a DL-Lite*$_{\mathcal{A},id}$ *TBox,* $\mathcal{A}$ *is a DL-Lite*$_{\mathcal{A},id}$ *ABox, and the following conditions are satisfied:*

(1) for each atomic role P*, if either* $(\mathsf{funct}\ P)$ *or* $(\mathsf{funct}\ P^-)$ *occur in* $\mathcal{T}$*, then* $\mathcal{T}$ *does not contain assertions of the form* $Q \sqsubseteq P$ *or* $Q \sqsubseteq P^-$*, where* Q *is a basic role;*
(2) for each atomic attribute U*, if* $(\mathsf{funct}\ U)$ *occurs in* $\mathcal{T}$*, then* $\mathcal{T}$ *does not contain assertions of the form* $U' \sqsubseteq U$*, where* U' *is an atomic attribute;*
(3) all concepts identified in $\mathcal{T}$ *are basic concepts, i.e., in each IdC* $(\mathsf{id}\ C\ \pi_1, \ldots, \pi_n)$ *of* $\mathcal{T}$*, the concept* C *is of the form* A*,* $\exists Q$*, or* $\delta(U)$*;*
(4) all concepts or value-domains appearing in the test relations in $\mathcal{T}$ *are of the form* A*,* $\exists Q$*,* $\delta(U)$*,* $\rho(U)$*,* $\top_D$*,* T_1*, ..., or* T_n*;*
(5) for each IdC α *in* $\mathcal{T}$*, every role or attribute that occurs (in either direct or inverse direction) in a path of* α *is not specialized in* $\mathcal{T}'$*, i.e., it does not appear in the right-hand side of assertions of the form* $Q \sqsubseteq Q'$ *or* $U \sqsubseteq U'$*.*

Intuitively, the conditions stated at points (1-2) (resp., (5)) say that, in *DL-Lite*$_{\mathcal{A},id}$ TBoxes, roles and attributes occurring in functionality assertions (resp., in paths of IdCs) cannot be specialized. All the above conditions are crucial for the tractability of reasoning in our logic.

The semantics of a *DL-Lite*$_{\mathcal{A},id}$ TBox is standard, except that it adopts the *unique name assumption*: for every interpretation $\mathcal{I}$, and distinct constants a, b, we have that $a^{\mathcal{I}} \neq b^{\mathcal{I}}$. Moreover, it takes into account the distinction between objects and values by partitioning the interpretation domain in two sets, containing objects and values, respectively. Note that the adoption of the unique name assumption in *DL-Lite*$_{\mathcal{A},id}$ makes it meaningless to use ABox assertions of the form $a = b$ and $a \neq b$, which instead occur in $\mathcal{ALCQI}b_{id}$ knowledge bases. Indeed, assertions of the first form cannot be satisfied by *DL-Lite*$_{\mathcal{A},id}$ interpretations, thus immediately making the knowledge base inconsistent, whereas assertions of the second form are always satisfied and are therefore implicit.

[6] We remind the reader that the identification constraints referred to in this paper are local.

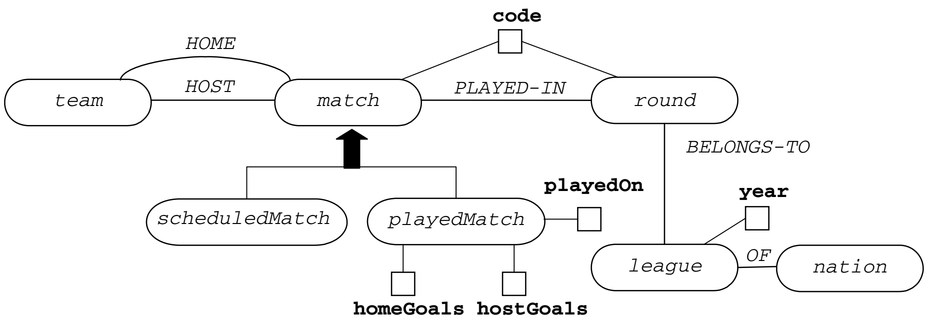


Fig. 6. Diagrammatic representation of the football leagues ontology

We finally recall a notable result given in [13], characterizing the complexity of query answering of UCQs over *DL-Lite*$_{\mathcal{A},id}$ knowledge bases. We remind the reader that AC$_0$ is the complexity class that corresponds to the complexity in the size of the data of evaluating a first-order (i.e., SQL) query over a relational database (see, e.g., [1]).

Theorem 1 ([13]). *Answering UCQs in DL-Lite*$_{\mathcal{A},id}$ *can be done in* AC$_0$ *with respect to the size of ABox.*

The above result is proved by showing that it is possible to reduce the query answering problem to the evaluation of a FOL query, directly translatable to SQL, over the database corresponding to the ABox assertions, thus exploiting standard commercial relational database technology.

Let us consider again the example on football leagues introduced in Section 2, and model it as a *DL-Lite*$_{\mathcal{A},id}$ TBox. By virtue of the characteristics of *DL-Lite*$_{\mathcal{A},id}$ we can now explicitly consider also attributes of concepts. In particular, we assume that when a scheduled match takes place, it is played in a specific date, and that for every match that has been played, the number of goals scored by the home team and by the host team are given. Note that different matches scheduled for the same round can be played in different dates. Also, we want to distinguish football championships on the basis of the nation and the year in which a championship takes place (e.g., the 2008 Spanish Liga). Finally, we assume that both matches and rounds have codes. In Figure 6, we show a schematic representation of (part of) the new ontology for the football leagues domain, whereas in Figure 7 the TBox assertions in *DL-Lite*$_{\mathcal{A},id}$ capturing the above aspects are shown. Note that, beside the new assertions involving attributes, Figure 7 lists all assertions given in Figure 3[7], which provide the $\mathcal{ALCQIb}_{id}$ TBox modeling of the football ontology, with the exception of the assertion *match* $\sqsubseteq$ *scheduledMatch* $\sqcup$ *playedMatch*. This is actually the price to pay to maintain reasoning tractable in *DL-Lite*$_{\mathcal{A},id}$, and in particular conjunctive query answering in AC$_0$. Indeed, the above assertion expresses the covering of the concept *match* with the concepts *scheduledMatch* and *playedMatch*, but as said in Section 3, the presence of covering assertions makes query answering coNP-hard in the size of the ABox.

[7] We have used $\exists R$ instead of $\exists R.\top$, and inclusions of the form $\top \sqsubseteq\ \leqslant 1\ R.\top$ are expressed as functional assertions of the form $(\mathsf{funct}\ R)$.

Inclusion assertions	
$\mathit{league} \sqsubseteq \exists \mathit{OF}$	$\mathit{playedMatch} \sqsubseteq \mathit{match}$
$\exists \mathit{OF} \sqsubseteq \mathit{league}$	$\mathit{scheduledMatch} \sqsubseteq \mathit{match}$
$\exists \mathit{OF}^- \sqsubseteq \mathit{nation}$	$\mathit{playedMatch} \sqsubseteq \neg \mathit{scheduledMatch}$
$\mathit{round} \sqsubseteq \exists \mathit{BELONGS\text{-}TO}$	
$\exists \mathit{BELONGS\text{-}TO} \sqsubseteq \mathit{round}$	$\mathit{league} \sqsubseteq \delta(\mathbf{year})$
$\exists \mathit{BELONGS\text{-}TO}^- \sqsubseteq \mathit{league}$	$\mathit{match} \sqsubseteq \delta(\mathbf{code})$
$\mathit{match} \sqsubseteq \exists \mathit{PLAYED\text{-}IN}$	$\mathit{round} \sqsubseteq \delta(\mathbf{code})$
$\exists \mathit{PLAYED\text{-}IN} \sqsubseteq \mathit{match}$	$\mathit{playedMatch} \sqsubseteq \delta(\mathbf{playedOn})$
$\exists \mathit{PLAYED\text{-}IN}^- \sqsubseteq \mathit{round}$	$\mathit{playedMatch} \sqsubseteq \delta(\mathbf{homeGoals})$
$\mathit{match} \sqsubseteq \exists \mathit{HOME}$	$\mathit{playedMatch} \sqsubseteq \delta(\mathbf{hostGoals})$
$\exists \mathit{HOME} \sqsubseteq \mathit{match}$	$\rho(\mathbf{playedOn}) \sqsubseteq \mathtt{xsd{:}date}$
$\exists \mathit{HOME}^- \sqsubseteq \mathit{team}$	$\rho(\mathbf{homeGoals}) \sqsubseteq \mathtt{xsd{:}nonNegativeInteger}$
$\mathit{match} \sqsubseteq \exists \mathit{HOST}$	$\rho(\mathbf{hostGoals}) \sqsubseteq \mathtt{xsd{:}nonNegativeInteger}$
$\exists \mathit{HOST} \sqsubseteq \mathit{match}$	$\rho(\mathbf{code}) \sqsubseteq \mathtt{xsd{:}positiveInteger}$
$\exists \mathit{HOST}^- \sqsubseteq \mathit{team}$	$\rho(\mathbf{year}) \sqsubseteq \mathtt{xsd{:}positiveInteger}$
Functional assertions	
(funct OF)	(funct $\mathbf{year}$)
(funct $\mathit{BELONGS\text{-}TO}$)	(funct $\mathbf{code}$)
(funct $\mathit{PLAYED\text{-}IN}$)	(funct $\mathbf{playedOn}$)
(funct HOME)	(funct $\mathbf{homeGoals}$)
(funct HOST)	(funct $\mathbf{hostGoals}$)
Identification constraints	
1. (id league OF, $\mathbf{year}$)	6. (id $\mathit{playedMatch}$ $\mathbf{playedOn}$, HOST)
2. (id round $\mathit{BELONGS\text{-}TO}$, $\mathbf{code}$)	7. (id $\mathit{playedMatch}$ $\mathbf{playedOn}$, HOME)
3. (id match $\mathit{PLAYED\text{-}IN}$, $\mathbf{code}$)	8. (id league $\mathbf{year}$, $\mathit{BELONGS\text{-}TO}^- \circ \mathit{PLAYED\text{-}IN}^- \circ \mathit{HOME}$)
4. (id match HOME, $\mathit{PLAYED\text{-}IN}$)	9. (id league $\mathbf{year}$, $\mathit{BELONGS\text{-}TO}^- \circ \mathit{PLAYED\text{-}IN}^- \circ \mathit{HOST}$)
5. (id match HOST, $\mathit{PLAYED\text{-}IN}$)	10. (id match HOME, HOST, $\mathit{PLAYED\text{-}IN} \circ \mathit{BELONGS\text{-}TO} \circ \mathbf{year}$)

Fig. 7. The TBox in $\mathit{DL\text{-}Lite}_{\mathcal{A},id}$ for the football leagues example

The identification constraints given in Figure 7 model the following aspects:

1. No nation has two leagues in the same year.
2. Within a league, the code associated to a round is unique.
3. Every match is identified by its code within its round.
4. A team is the home team of at most one match per round.
5. As above for the host team.
6. No home team participates in different played matches in the same date
7. As above for the host team.
8. No home team plays in different leagues in the same year.
9. As above for the host team.
10. No pair (home team, host team) plays different matches in the same year.

5 Data Integration with $\mathit{DL\text{-}Lite}_{\mathcal{A},id}$

In this section, we illustrate a specific proposal of data integration system based on $\mathit{DL\text{-}Lite}_{\mathcal{A},id}$, by describing the three components of the system. The choice for the languages used in the three components is tailored towards the goal of an optimal trade-off between expressive power and complexity. After the description of the three components, we briefly illustrate the algorithm for answering queries in our approach, and we discuss its computational complexity.

5.1 The Global Schema

As said before, one of the basic characteristics of the approach to data integration advocated in this paper is that the global schema represents the conceptual model of the domain of interest, rather than a mere description of a unified view of the source data. We have also discussed the advantages of expressing the conceptual model in terms of a DL. Finally, we have seen in the previous section that *DL-Lite*$_{\mathcal{A},id}$ represents a valuable choice as a formalism for expressing the global schema of a data integration system. Therefore, in our approach to data integration, the global schema is expressed in terms of a *DL-Lite*$_{\mathcal{A},id}$ TBox. It is interesting to observe that most of the properties of class diagrams discussed in Section 3 can indeed be expressed in *DL-Lite*$_{\mathcal{A},id}$. The only modeling construct that cannot be fully represented is covering. In other words, generalizations expressible in *DL-Lite*$_{\mathcal{A},id}$ are not complete. Also, functional associations cannot be specialized.

5.2 The Source Schema

If $\mathcal{J} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$ is a data integration system in our approach, $\mathcal{S}$ is assumed to be a flat relational database schema, representing the schemas of all the data sources. Actually, this is not a limitation of the system, since the source schema can always be thought of as the schema managed by a relational data federation tool. Specifically, we assume that a data federation tool is in charge of interacting with the data sources, presenting them as a single relational database schema. Such a schema is obtained by wrapping physical sources, possibly heterogeneous, and not necessarily in relational format. Furthermore, the data federation tool is in charge of answering queries formulated over the source schema, by suitably transforming such queries, forwarding them to the right sources, and finally combining the single results into the overall answer. In other words, the data federation tool makes the whole system independent from the physical nature of the sources, by providing a logical representation of them (physical independence), whereas the other components of the system make all the logical aspects transparent to the user, by maintaining the conceptual global schema separate from the logical federated schema, and connecting them via suitable mappings (logical independence).

5.3 The Mapping

The mappings in our approach establish the relationship between the source schema and the global schema, thus specifying how data stored at the sources are linked to the instances of the concepts and the roles in the global schema. More specifically, we follow the *GAV (global-as-view)* approach for specifying mappings, which requires to describe the meaning of every element of the global schema by associating to it a view over the sources. The dual approach, called LAV (local-as-view), would require the sources to be defined as views over the global schema.

Moreover, our mapping specification language takes suitably into account the *impedance mismatch* problem, i.e., the mismatch between the way in which data is (and can be) represented in a data source, and the way in which the corresponding information is rendered through the global schema.

The mapping assertions keep data value constants separate from object identifiers, and construct identifiers as (logic) terms over data values. More precisely, object identifiers in our approach are *terms* of the form $f(d_1, \ldots, d_n)$, called *object terms*, where f is a function symbol of arity $n > 0$, and $d_1, \ldots, d_n$ are data values stored at the sources. Note that this idea traces back to the work done in deductive object-oriented databases [24].

We detail below the above ideas. The mapping component is specified through a set of mapping assertions, each of the form

$$\Phi(\boldsymbol{v}) \rightsquigarrow G(\boldsymbol{w})$$

where

– $\Phi(\boldsymbol{v})$, called the *body* of the mapping, is a first-order logic (FOL) query of arity $n > 0$, with distinguished variables $\boldsymbol{v}$, over the source schema $\mathcal{S}$ (we will write such query in the SQL syntax), and
– $G(\boldsymbol{w})$, called the *head*, is an atom where G can be an atomic concept, an atomic role, or an atomic attribute occurring in the global schema $\mathcal{G}$, and $\boldsymbol{w}$ is a sequence of terms.

We define three different types of mapping assertions:

– *Concept mapping assertions*, in which the head is a unary atom of the form $A(f(\boldsymbol{v}))$, where A is an atomic concept and f is a function symbol of arity n;
– *Role mapping assertions*, in which the head is a binary atom of the form $P(f_1(\boldsymbol{v}'), f_2(\boldsymbol{v}''))$, where P is an atomic role, f_1 and f_2 are function symbols of arity $n_1, n_2 > 0$, and $\boldsymbol{v}'$ and $\boldsymbol{v}''$ are sequences of variables appearing in $\boldsymbol{v}$;
– *Attribute mapping assertions*, in which the head is a binary atom of the form $U(f(\boldsymbol{v}'), v'' : T_i)$, where U is an atomic attribute, f is a function symbol of arity $n' > 0$, $\boldsymbol{v}'$ is a sequence of variables appearing in $\boldsymbol{v}$, v'' is a variable appearing in $\boldsymbol{v}$, and T_i is an RDF data type.

In words, such mapping assertions are used to map source relations (and the tuples they store), to concepts, roles, and attributes of the ontology (and the objects and the values that constitute their instances), respectively. Note that an attribute mapping also specifies the type of values retrieved from the source database, in order to guarantee coherency of the system.

We conclude this section with an example of mapping assertions, referring again to the football domain. Suppose that the source schema contains the relational table `TABLE(mcode,league,round,home,host)`, where a tuple (m, l, r, h_1, h_2) with $l > 0$ represents a match with code m of league l and round r, and with home team h_1 and host team h_2. If we want to map the tuples from the table `TABLE` to the global schema shown in Figure 7, the mapping assertions might be as shown in Figure 8. M_1 is a concept mapping assertion that selects from `TABLE` the code and the round of matches (only for the appropriate tuples), and then uses such data to build instances of the concept *match*, using the function symbol **m**. M_2 is an attribute mapping assertion that is used to "populate" the attribute **code** for the objects that are instances of *match*. Finally, M_3 is a role mapping assertion relating `TABLE` to the atomic role *PLAYED-IN*,

M_1:	`SELECT T.mcode, T.round, T.league FROM TABLE T WHERE T.league > 0`	$\leadsto$	*match*(**m**(`T.mcode`, `T.round`, `T.league`))
M_2:	`SELECT T.mcode, T.round, T.league FROM TABLE T WHERE T.league > 0`	$\leadsto$	**code**(**m**(`T.mcode`, `T.round`, `T.league`), `T.mcode` : `xsd:string`)
M_3:	`SELECT T.mcode, T.round, T.league FROM TABLE T WHERE T.league > 0`	$\leadsto$	*PLAYED-IN*(**m**(`T.mcode`, `T.round`, `T.league`), **r**(`T.round`, `T.league`))

Fig. 8. Example of mapping assertions

where instances of *round* are denoted by means of the function symbol **r**. We notice that in the mapping assertion M_2, the mapping designer had to specify a correct *DL-Lite*$_{\mathcal{A},id}$ data type for the values extracted from the source.

We point out that, during query answering, the body of each mapping assertion is never really evaluated in order to extract values from the sources to build instances of the global schema, but rather it is used to unfold queries posed over the global schema, rewriting them into queries posed over the source schema. We discuss this aspect next.

5.4 Query Answering

We sketch here our query answering technique (more details can be found in [30,10]). Consider a data integration system $\mathcal{J} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$ and a database D for $\mathcal{S}$, and assume that $\mathcal{J}$ is satisfiable with respect to D, i.e., $sem_D(\mathcal{J}) \neq \emptyset$ (cf. Section 2.1).

We start with the following observation. Suppose we evaluate (over D) the queries in the left-hand sides of the mapping assertions, and we materialize accordingly the corresponding assertions in the right-hand sides. This would lead to a set of ground assertions, denoted by $\mathcal{A}^{\mathcal{M},D}$, that can be considered as a *DL-Lite*$_{\mathcal{A},id}$ ABox. It can be shown that query answering over $\mathcal{J}$ and D can be reduced to query answering over the *DL-Lite*$_{\mathcal{A},id}$ knowledge base constituted by the TBox $\mathcal{G}$ and the ABox $\mathcal{A}^{\mathcal{M},D}$. However, due to the materialization of $\mathcal{A}^{\mathcal{M},D}$, the query answering algorithm resulting from this approach would be polynomial in the size of D. On the contrary, our idea is to avoid any ABox materialization, but rather answer Q by reformulating it into a new query that can be afterwards evaluated directly over the database D. The resulting query answering algorithm is much more efficient than the one sketched above, and is constituted by four steps, which are called *rewriting*, *filtering*, *unfolding*, and *evaluation*, and are described in the following.

Rewriting. Given a UCQ Q over a data integration system $\mathcal{J} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$, and a source database D for $\mathcal{J}$, the rewriting step computes a new UCQ Q_1 over $\mathcal{J}$, where the assertions of $\mathcal{G}$ are compiled in. In computing the rewriting, only inclusion assertions of the form $B_1 \sqsubseteq B_2$, $Q_1 \sqsubseteq Q_2$, and $U_1 \sqsubseteq U_2$ are taken into account, where B_i, Q_i, and U_i, with $i \in \{1, 2\}$, are a basic concept, a basic role, and an atomic attribute, respectively. Intuitively, the query Q is rewritten according to the knowledge specified in $\mathcal{G}$ that is relevant for answering Q, in such a way that the rewritten query Q_1 is such that $Q_1^{\langle \emptyset, \mathcal{S}, \mathcal{M} \rangle, D} = Q^{\mathcal{J}, D}$, i.e., the rewriting allows to get rid of $\mathcal{G}$.

We refer to [30,12] for a formal description of the query rewriting algorithm and for a proof of its soundness and completeness. We only notice here that the rewriting procedure does not depend on the source database D, runs in polynomial time in the size of $\mathcal{G}$, and returns a query Q_1 whose size is at most exponential in the size of Q.

Filtering. Let Q_1 be the UCQ produced by the rewriting step above. In the filtering step we take care of a particular problem that the disjuncts, i.e., conjunctive queries, in Q_1 might have. Specifically, a conjunctive query cq is called *ill-typed* if it has at least one join variable x appearing in two incompatible positions in cq, i.e., such that the TBox $\mathcal{G}$ of our data integration system logically implies that x is both of type T_i, and of type T_j, with $T_i \neq T_j$ (remember that in *DL-Lite*$_{\mathcal{A},id}$ data types are pairwise disjoint). The purpose of the filtering step is to remove from the UCQ Q_1 all the ill-typed conjunctive queries. Intuitively, such a step is needed because the query Q_1 has to be unfolded and then evaluated over the source database D (cf. the next two steps of the query answering algorithm, described below). These last two steps, performed for an ill-typed conjunctive query might produce incorrect results.

Unfolding. Given the UCQ Q_2 over $\mathcal{J}$ computed by the filtering step, the unfolding step computes, by using logic programming techniques, an SQL query Q_3 over the source schema $\mathcal{S}$, that possibly returns object terms. It can be shown [30] that Q_3 is such that $Q_3^D = Q_2^{\langle \emptyset, \mathcal{S}, \mathcal{M} \rangle, D}$, i.e., unfolding allows us to get rid of $\mathcal{M}$. Moreover, the unfolding procedure does not depend on D, runs in polynomial time in the size of $\mathcal{M}$, and returns a query whose size is polynomial in the size of Q_2.

Evaluation. The evaluation step consists in simply delegating the evaluation of the SQL query Q_3, produced by the unfolding step, to the data federation tool managing the data sources. Formally, such a tool returns the set Q_3^D, i.e., the set of tuples obtained from the evaluation of Q_3 over D.

5.5 Correctness and Complexity of Query Answering

It can be shown that the query answering procedure described above correctly computes the certain answers to UCQs. Based on the computational properties of such an algorithm, we can then characterize the complexity of our query answering method.

Theorem 2. *Let $\mathcal{J} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$ be a data integration system, and D a source database for $\mathcal{J}$. Answering a UCQ over $\mathcal{J}$ with respect to D can be reduced to the evaluation of an SQL query over D, and can be done in* AC_0 *in the size of D.*

In other words, the above theorem says that UCQs in our approach are FO-rewritable.

Finally, we remark that, as we said at the beginning of this section, we have assumed that the data integration system $\mathcal{J}$ is consistent with respect to the database D, i.e., $sem_D(\mathcal{J})$ is non-empty. Notably, it can be shown that all the machinery we have devised for query answering can also be used for checking consistency of $\mathcal{J}$ with respect to D. Therefore, checking consistency can also be reduced to sending appropriate SQL queries to the source database [30,13].

6 Extending the Data Integration Framework

We now analyze the possibility of extending the data integration setting presented above without affecting the complexity of query answering. In particular, we investigate possible extensions for the language for expressing the global schema, the language for expressing the mappings, and the language for expressing the source schema.

We start by dealing with extending the global schema language. There are two possible ways of extending *DL-Lite*$_{\mathcal{A},id}$. The first one corresponds to a proper language extension, i.e., adding new DL constructs to *DL-Lite*$_{\mathcal{A},id}$, while the second one consists of changing/strengthening the semantics of the formalism.

Concerning language extensions, the results in [11] show that it is not possible to add any of the usual DL constructs to *DL-Lite*$_{\mathcal{A},id}$ while keeping the data complexity of query answering within AC_0. This means that *DL-Lite*$_{\mathcal{A},id}$ is essentially the most expressive DL allowing for data integration systems where query answering is FO-rewritable.

Concerning the possibility of strengthening the semantics, we briefly analyze the consequences of removing the *unique name assumption* (UNA), i.e., the assumption that, in every interpretation of a data integration system, two distinct object terms and two distinct value constants denote two different domain elements. Unfortunately, this leads query answering out of LOGSPACE, and therefore, this leads to loosing FO-rewritability of queries.

Theorem 3 ([10]). *Let $\mathcal{J} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$ be a DL-Lite$_{\mathcal{A},id}$ data integration system extended by removing the UNA, and D a database for $\mathcal{S}$. Computing the certain answers to a query constituted by a single atom in $\mathcal{J}$ with respect to D is* NLOGSPACE*-hard in the size of D.*

Next we consider extensions to the mapping language. The possibility of extending the language used to express the mapping has been analyzed in [10], which considers the so-called GLAV mappings, i.e., assertions that relate conjunctive queries over the sources to conjunctive queries over the global schema. Such assertions are therefore an extension of both GAV and LAV mappings. Unfortunately, even with LAV mappings only, it has been shown that instance checking and query answering are no more in LOGSPACE with respect to data complexity. Thus, with LAV mappings, we again loose FO-rewritability of UCQs.

Theorem 4 ([10]). *Let $\mathcal{J} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$ be a DL-Lite$_{\mathcal{A},id}$ data integration system extended with LAV mapping assertions, and D a database for $\mathcal{S}$. Computing the certain* *answers to a query constituted by a single atom in $\mathcal{J}$ with respect to D is* NLOGSPACE-*hard in the size of D.*

Finally, we consider the possibility of handling source schemas beyond the relational model. The data integration architecture referred to in this paper assumes to deal with relational sources, managed by a relational data federation tool. It is not hard to see, however, that all the results mentioned here apply also if we consider federation tools that provide a representation of the data at the sources according to a different data model (e.g., XML). Obviously, depending on the specific data model adopted by the data federation tool, we have to resort to a suitable query language for expressing the source queries appearing in the mapping assertions. To adhere to the framework adopted in this paper, the only constraint imposed on the query language (that is trivially satisfied by virtually all query languages used in practice) is that it is able to extract tuples of values from the sources.

7 Discussion and Conclusions

Starting from the late 90s, research in data integration has mostly focused on declarative approaches (as opposed to procedural ones) [32,26], such as the one advocated here. The GAV approach for specifying mappings has been proposed e.g., in [15,20,31,22], while the LAV approach is at the basis of the work in [25,18,14].

Although in the present work we make use of GAV mappings, the presence of constraints expressed in a rich ontology language in the global schema, makes query answering in our setting more similar to what is carried out in LAV data integration systems rather than in GAV systems. Indeed, while in general GAV systems have been realized as (simple) hierarchies of wrappers and mediators, query answering in LAV can be considered a form of reasoning in the presence of incomplete information, and thus significantly more complex. Early systems based on this approach, like Information Manifold [28,29], or INFOMASTER [21,19], have implemented algorithms [28] for rewriting queries using views (where the views are the ones specified through the CQs in the mappings). The relationship between LAV and GAV data integration systems is explored in [7], where it is indeed shown that a LAV system can be converted into a GAV one by introducing suitable inclusion dependencies in the global schema. If no functionality assertions are present in the global schema, such inclusion dependencies can then be dealt with in a way similar to what is done here for concept and role inclusions in *DL-Lite*$_{\mathcal{A},id}$. Note that this is no longer possible, instead, in the presence of functionality assertions.

The approach illustrated in this paper has been implemented in a prototype system called MASTRO-I (see [13]). We conclude the paper by mentioning some aspects that are important for the problem of semantic data integration, but that have not been addressed yet in the development of the system.

A first important point is handling inconsistencies in the data, possibly using a declarative, rather than an ad-hoc procedural approach. An interesting proposal is the one of the INFOMIX system [27] for the integration of heterogeneous data sources (e.g., relational, XML, HTML) accessed through a relational global schema with powerful forms of integrity constraints. The query answering technique proposed in such a system is based on query rewriting in Datalog enriched with negation and disjunction, under stable model semantics [8,23].

A second interesting issue for further work is looking at "write-also" data integration tools. Indeed, while the techniques presented in this paper provide support for answering queries posed to the data integration system, it could be of interest to also deal with updates expressed on the global schema (e.g., according to the approach described in [16,17]). The most challenging issue to be addressed in this context is to design mechanisms for correctly reformulating an update expressed over the ontology into a series of insert and delete operations on the data sources.

Acknowledgements. This research has been partially supported by the IP project OntoRule (ONTOlogies meet Business RULEs ONtologiES), funded by the EC under ICT Call 3 FP7-ICT-2008-3, contract number FP7-231875, by project DASIbench (Data and Service Integration workbench), funded by IBM through a Faculty Award grant, and by the MIUR FIRB 2005 project "Tecnologie Orientate alla Conoscenza per Aggregazioni di Imprese in Internet" (TOCAI.IT).

References

1. Abiteboul, S., Hull, R., Vianu, V.: Foundations of Databases. Addison Wesley Publ. Co., Reading (1995)
2. Baader, F., Calvanese, D., McGuinness, D., Nardi, D., Patel-Schneider, P.F. (eds.): The Description Logic Handbook: Theory, Implementation and Applications. Cambridge University Press, Cambridge (2003)
3. Berardi, D., Calvanese, D., De Giacomo, G.: Reasoning on UML class diagrams. Artificial Intelligence 168(1–2), 70–118 (2005)
4. Bernstein, P.A., Giunchiglia, F., Kementsietsidis, A., Mylopoulos, J., Serafini, L., Zaihrayeu, I.: Data management for peer-to-peer computing: A vision. In: Proc. of the 5th Int. Workshop on the Web and Databases, WebDB 2002 (2002)
5. Bernstein, P.A., Haas, L.: Informaton integration in the enterprise. Communications of the ACM 51(9), 72–79 (2008)
6. Brodie, M.L., Mylopoulos, J., Schmidt, J.W. (eds.): On Conceptual Modeling: Perspectives from Artificial Intelligence, Databases, and Programming Languages. Springer, Heidelberg (1984)
7. Calì, A., Calvanese, D., De Giacomo, G., Lenzerini, M.: On the expressive power of data integration systems. In: Spaccapietra, S., March, S.T., Kambayashi, Y. (eds.) ER 2002. LNCS, vol. 2503, pp. 338–350. Springer, Heidelberg (2002)
8. Calì, A., Lembo, D., Rosati, R.: Query rewriting and answering under constraints in data integration systems. In: Proc. of the 18th Int. Joint Conf. on Artificial Intelligence (IJCAI 2003), pp. 16–21 (2003)
9. Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Poggi, A., Rosati, R.: Linking data to ontologies: The description logic *DL-Lite*$_A$. In: Proc. of the 2nd Int. Workshop on OWL: Experiences and Directions (OWLED 2006). CEUR Electronic Workshop Proceedings, vol. 216 (2006), http://ceur-ws.org/
10. Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Poggi, A., Rosati, R., Ruzzi, M.: Data integration through *DL-Lite*$_A$ ontologies. In: Schewe, K.-D., Thalheim, B. (eds.) SDKB 2008. LNCS, vol. 4925, pp. 26–47. Springer, Heidelberg (2008)

11. Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Rosati, R.: Data complexity of query answering in description logics. In: Proc. of the 10th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR 2006), pp. 260–270 (2006)
12. Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Rosati, R.: Tractable reasoning and efficient query answering in description logics: The *DL-Lite* family. J. of Automated Reasoning 39(3), 385–429 (2007)
13. Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Rosati, R.: Path-based identification constraints in description logics. In: Proc. of the 11th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR 2008), pp. 231–241 (2008)
14. Calvanese, D., De Giacomo, G., Lenzerini, M., Nardi, D., Rosati, R.: Data integration in data warehousing. Int. J. of Cooperative Information Systems 10(3), 237–271 (2001)
15. Carey, M.J., Haas, L.M., Schwarz, P.M., Arya, M., Cody, W.F., Fagin, R., Flickner, M., Luniewski, A., Niblack, W., Petkovic, D., Thomas, J., Williams, J.H., Wimmers, E.L.: Towards heterogeneous multimedia information systems: The Garlic approach. In: Proc. of the 5th Int. Workshop on Research Issues in Data Engineering – Distributed Object Management (RIDE-DOM 1995), pp. 124–131. IEEE Computer Society Press, Los Alamitos
16. De Giacomo, G., Lenzerini, M., Poggi, A., Rosati, R.: On the update of description logic ontologies at the instance level. In: Proc. of the 21st Nat. Conf. on Artificial Intelligence (AAAI 2006), pp. 1271–1276 (2006)
17. De Giacomo, G., Lenzerini, M., Poggi, A., Rosati, R.: On the approximation of instance level update and erasure in description logics. In: Proc. of the 22nd Nat. Conf. on Artificial Intelligence (AAAI 2007), pp. 403–408 (2007)
18. Duschka, O.M., Genesereth, M.R.: Answering recursive queries using views. In: Proc. of the 16th ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS 1997), pp. 109–116 (1997)
19. Duschka, O.M., Genesereth, M.R., Levy, A.Y.: Recursive query plans for data integration. J. of Logic Programming 43(1), 49–73 (2000)
20. Garcia-Molina, H., Papakonstantinou, Y., Quass, D., Rajaraman, A., Sagiv, Y., Ullman, J.D., Vassalos, V., Widom, J.: The TSIMMIS approach to mediation: Data models and languages. J. of Intelligent Information Systems 8(2), 117–132 (1997)
21. Genereseth, M.R., Keller, A.M., Duschka, O.M.: Infomaster: An information integration system. In: Proc. of the ACM SIGMOD Int. Conf. on Management of Data, pp. 539–542 (1997)
22. Goh, C.H., Bressan, S., Madnick, S.E., Siegel, M.D.: Context interchange: New features and formalisms for the intelligent integration of information. ACM Trans. on Information Systems 17(3), 270–293 (1999)
23. Grieco, L., Lembo, D., Ruzzi, M., Rosati, R.: Consistent query answering under key and exclusion dependencies: Algorithms and experiments. In: Proc. of the 14th Int. Conf. on Information and Knowledge Management (CIKM 2005), pp. 792–799 (2005)
24. Hull, R.: A survey of theoretical research on typed complex database objects. In: Paredaens, J. (ed.) Databases, pp. 193–256. Academic Press, London (1988)
25. Kirk, T., Levy, A.Y., Sagiv, Y., Srivastava, D.: The Information Manifold. In: Proceedings of the AAAI 1995 Spring Symp. on Information Gathering from Heterogeneous, Distributed Enviroments, pp. 85–91 (1995)
26. Lenzerini, M.: Data integration: A theoretical perspective. In: Proc. of the 21st ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS 2002), pp. 233–246 (2002)
27. Leone, N., Eiter, T., Faber, W., Fink, M., Gottlob, G., Greco, G., Kalka, E., Ianni, G., Lembo, D., Lenzerini, M., Lio, V., Nowicki, B., Rosati, R., Ruzzi, M., Staniszkis, W., Terracina, G.: The INFOMIX system for advanced integration of incomplete and inconsistent data. In: Proc. of the ACM SIGMOD Int. Conf. on Management of Data, pp. 915–917 (2005)

28. Levy, A.Y., Rajaraman, A., Ordille, J.J.: Querying heterogenous information sources using source descriptions. In: Proc. of the 22nd Int. Conf. on Very Large Data Bases, VLDB 1996 (1996)
29. Levy, A.Y., Srivastava, D., Kirk, T.: Data model and query evaluation in global information systems. J. of Intelligent Information Systems 5, 121–143 (1995)
30. Poggi, A., Lembo, D., Calvanese, D., De Giacomo, G., Lenzerini, M., Rosati, R.: Linking data to ontologies. In: Spaccapietra, S. (ed.) Journal on Data Semantics X. LNCS, vol. 4900, pp. 133–173. Springer, Heidelberg (2008)
31. Tomasic, A., Raschid, L., Valduriez, P.: Scaling access to heterogeneous data sources with DISCO. IEEE Trans. on Knowledge and Data Engineering 10(5), 808–823 (1998)
32. Ullman, J.D.: Information integration using logical views. In: Afrati, F.N., Kolaitis, P.G. (eds.) ICDT 1997. LNCS, vol. 1186, pp. 19–40. Springer, Heidelberg (1996)

Clio: Schema Mapping Creation and Data Exchange

Ronald Fagin[1], Laura M. Haas[1], Mauricio Hernández[1], Renée J. Miller[2], Lucian Popa[1], and Yannis Velegrakis[3]

[1] IBM Almaden Research Center, San Jose, CA 95120, USA
[2] University of Toronto, Toronto ON M5S2E4, Canada
[3] University of Trento, 38100 Trento, Italy
fagin@almaden.ibm.com, laura@almaden.ibm.com, mauricio@almaden.ibm.com, miller@cs.toronto.edu, lucian@almaden.ibm.com, velgias@disi.unitn.eu

Abstract. The Clio project provides tools that vastly simplify information integration. Information integration requires data conversions to bring data in different representations into a common form. Key contributions of Clio are the definition of non-procedural *schema mappings* to describe the relationship between data in heterogeneous schemas, a new paradigm in which we view the mapping creation process as one of query discovery, and algorithms for automatically generating queries for data transformation from the mappings. Clio provides algorithms to address the needs of two major information integration problems, namely, *data integration* and *data exchange*. In this chapter, we present our algorithms for both schema mapping creation via query discovery, and for query generation for data exchange. These algorithms can be used in pure relational, pure XML, nested relational, or mixed relational and nested contexts.

1 Introduction

We present a retrospective on key contributions of the Clio project, a joint project between the IBM Almaden Research Center and the University of Toronto begun in 1999. Clio's goal is to radically simplify information integration, by providing tools that help in automating and managing one challenging piece of that problem: the conversion of data between representations. Clio pioneered the use of *schema mappings*, specifications that describe the relationship between data in two heterogeneous schemas. From this high-level, non-procedural representation, it can automatically generate either a view, to reformulate queries against one schema into queries on another for *data integration*, or code, to transform data from one representation to the other for *data exchange*. In this chapter, we focus on two key components of Clio: the creation of mappings between heterogeneous schemas, and their use for the implementation of data exchange.

1.1 Schema Mapping

Schema mappings are fundamental for a number of important information integration problems [9] including data integration, data exchange, peer-to-peer data sharing, schema integration and schema evolution. Applications are typically limited to handling information with a specific schema, so they rely on systems that can create and use mappings to transform data from one representation to another.

A fundamental requirement for Clio is that it make no assumption about the relationship between the schemas or how they were created. In particular, we do not assume that either of the schemas is a global or mediator schema, nor that one schema is a view (global or otherwise) over the other. This implies that both schemas may contain data not represented in the other, and that both may have their own constraints.

This requirement to map independently created schemas has a strong impact on our mapping language, as we need one that is more general than those used in traditional schema integration [6] or in mediator systems such as TSIMMIS [16] or Information Manifold [34].

A second requirement is that we be able to map between relational schemas and nested schemas (for example, XML schemas). As XML emerged as a common standard for exchanging data, an early motivating application for our work was publishing legacy relational data in XML. This often requires relational data to be placed into a predefined XML schema (defined, e.g., by a standards committee to permit meaningful exchange within a specific domain). However, for other applications including schema evolution, data warehousing, and data federation, we also need to be able to map data between different relational schemas and between any combination of nested and relational schemas.

A third requirement is that we be able to create and use mappings at different levels of granularity. For some applications and some schemas, it may be sufficient to create fine-grained mappings between individual components (for example, between attributes or elements to translate gross salary in francs to net salary in dollars). For others, mappings between broader concepts are required (for example, between the order concept in one billing application with that used by another). And for other applications, we may need to map full documents (for example, map every company's carbon emission data expressed in a schema suitable for the European Union Emission Trading Scheme to a schema designed for the Portable Emissions Measurement Systems standard).

Finally, we want our mapping creation algorithms to be incremental. There are many motivations for this. First, for many tasks, complete mapping of one entire schema to another is not the goal. It may suffice to map a single concept to achieve the desired interoperability. Second, we want a tool that gives users (or systems) with only partial knowledge of the schemas, or limited resources, useful (and usable) mappings despite their incomplete knowledge or resources. We hope that incomplete mappings can help them in understanding the schemas and data better, and that the mappings can be refined over time as need arises, for example, as new data appears, or the application needs change. This particular aspect of our approach was explored in more detail in our work on data-driven mapping refinement [51] and in work on mapping debugging [3], but will not be emphasized in this chapter. This ability to evolve mappings incrementally has more recently been coined *pay-as-you-go* [23].

Clio mappings assume that we are given two schemas and that we would like to map data from the first to the second. We refer to the first schema as a *source* schema, and the second as a *target schema*. In practice, this meets most application needs, as most require only a uni-directional flow of data. For example, one common use of mappings is in query reformulation, commonly referred to as *data integration* [33], where queries on a target schema are reformulated, using the mappings, into queries on a source schema. For applications requiring bi-directional mapping, mappings are created in both directions.

1.2 Implementing Data Exchange

Another common use of mappings is for *data exchange* where the goal is to create a target instance that reflects the source instance as accurately as possible [19]. Since the target is materialized, queries on the target schema can be answered directly without query reformulation. At the time we started Clio, data integration had been widely studied, but work on data exchange was quite dated. Foundational systems like Express [46,47] did data exchange for mappings which were much less expressive than those needed to map arbitrary schemas. There were no systems that performed data exchange for the general mappings we strove to create.

For independent schemas, because the schemas may represent overlapping but distinct sets of concepts, a schema mapping may relate a source instance with many *possible* target instances. As a result, we have a fundamentally new problem: given a schema mapping, determine which possible target instance is the best one to use for data exchange. At the time of our first results [38,43], this problem had not yet been formalized. Hence, in Clio, we made some intuitive decisions that were later formalized into a theory for data exchange [19]. In this chapter, we discuss this intuition, and how it corresponds to the later theory. We also discuss some important systems issues not covered by the theory. Specifically, we consider how to create a *data exchange program* from a schema mapping. Due to the requirements for schema mapping laid out above, we choose to produce executable queries for data exchange. A schema mapping is a declarative specification of how an instance of a source schema corresponds to possibly (infinitely) many target instances, and from this we choose a best target instance to materialize. Hence, our data exchange queries, when executed on a source instance, will generate this one chosen target instance.

The origins of this chapter first appeared in Miller et al. [38] and Popa et al. [43]. This chapter also includes details, originally from Velegrakis [48], of our data exchange implementation. The requirements outlined above force us to accommodate various runtime environments. In this chapter, we discuss how to generate data exchange queries in SQL, XQuery or XSLT.

The chapter is organized as follows. Section 2 introduces a motivating example and describes the problem of schema mapping generation and the problem of data exchange. Section 3 presents the data model we will use for representing both relational and nested relational schemas along with our schema mapping formalism. Section 4 presents our algorithm for generating schema mappings. Section 5 presents our algorithm for data exchange (mapping code generation). We present a discussion and analysis of our algorithms in Section 6, describe the related work in Section 7 and then conclude.

2 A Motivating Example

To motivate our approach, we first walk through an example explaining the intuition behind our mapping creation algorithm, and highlighting some of its features. We then extend our example to illustrate how schema mappings can be used to generate queries for data exchange, and describe the key innovations in that algorithm.

2.1 Schema Mapping Creation

Schema. Consider a data source with information about companies and grants. The structure of its data is described by the Schema `S`, illustrated in Figure 1. It is a relational schema containing three tables, `companies`, `grants`, and `contacts`, presented in a nested relational representation that we use to model both relational and XML schemas. It contains a set of grants (`grants`), each consisting of a grant identifier (`gid`), a recipient (`recipient`), its amount (`amount`), its supervisor (`supervisor`) and its manager (`manager`). The `recipient` is actually the name of the company that received the grant. For each company, the database stores its name (`name`), address (`address`) and the year it was founded (`year`). Similarly, the supervisor and manager are references to some contact information, which consists of an identifier (`cid`), an email (`email`) and a phone number (`phone`). The curved lines f_1, f_2 and f_3 in the figure represent referential constraints specified as part of the schema. For example, f_1 may be a foreign key, or simply an inclusion dependency, stating that values in `grants.recipient` must also appear in `companies.name`.

Consider a second schema T, as illustrated on the right-hand side of Figure 1. It records the funding (`fundings`) that an organization (`organizations`) receives, nested within the organization record. The amount of each funding (`budget`) is kept in the `finances` record along with a contact phone number (`phone`). The target may be an XML schema containing a referential constraint in the form of a *keyref* definition (f_4).

Correspondences. To begin to understand the relationship between the schemas, we may invoke a schema matcher to generate a set of element correspondences (or matchings). Alternatively, we could ask a user (for example, a data designer or administrator familiar with the schemas) to draw lines between

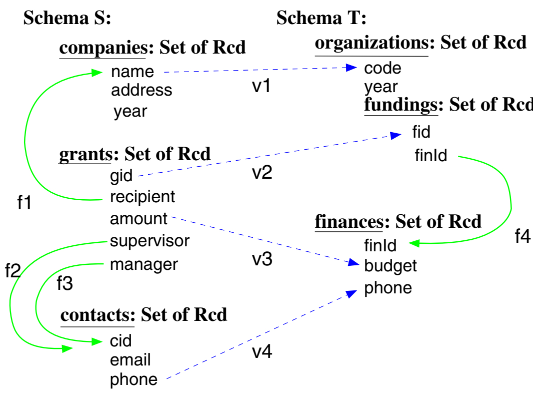


Fig. 1. A source and a target schema in a mapping scenario

elements that should contain related data. In our example, the dashed arrows between the elements of the two schemas in Figure 1 represent a set of matchings or correspondences. The line v_1 indicates (informally) that what is called a company `name` in the first schema, is referred to as an organization `code` in the second. In contrast, both schemas have an element `year`, but the data administrator (or matching tool) has specified no arrow between them. That may be because there is reason to believe that these elements do not represent the same concept. For instance, element `year` in the first schema may represent the time the company was founded, while in the second it may represent the time the company had its initial public offer (IPO).

Our approach is agnostic to how correspondences are created, whether manually or (semi-)automatically, but is cognizant that matchings are often incomplete, and sometimes incorrect. Hence, we have developed techniques for incrementally modifying mappings as correspondences change [49]. To keep our example simple, we assume that correspondences v_1, v_2, v_3, v_4 are correct.

Mappings. One way a correspondence can be interpreted is that the target schema element should be populated with values from the source schema element. This can be formally expressed using an inter-schema inclusion dependency or more generally through a source-to-target *tuple generating dependency*, (*tgd*) [7]. A tgd representing correspondence v_1 of Figure 1 is shown below (the nested set `fundings` which is represented by the variable F inside `organizations` will be explained below).

$$\forall \underline{n}, d, y, \texttt{companies}(\underline{n}, d, y) \;\rightarrow\; \exists y', F\; \texttt{organizations}(\underline{n}, y', F) \qquad (1)$$

This simple mapping states that for each `companies` tuple, there must be an `organizations` tuple whose code is the same as the `companies.name`; this is represented by the shared variable n, which carries source data to the target. As a convention, when writing tgds, we underline all the variables that appear in both the left-hand side and the right-hand side of the implication. This target tuple must have a value for the `year` attribute, but the mapping does not specify what this value should be (this is represented by the existential variable y'). Similarly, we could write simple tgds to express the other three correspondences.

If our application only requires information about organizations (and not about `fundings` or `finances`), then we can stop the mapping generation here. This simple element-to-element mapping can be used for data exchange or for data integration [52]. However, if the user is interested in mapping more data, we can continue the mapping generation using the other correspondences from Figure 1 to map additional concepts.

Associations. Correspondences alone do not specify how individual data values should be connected in the target. For example, in the target, funding information is nested inside organizations. This nesting indicates that there is a semantic association between organization and funding records. This may represent organizations and the funding that the organization has received, or possibly organizations and the funding that they have given to others. The semantics is not specified precisely in the schema, but it is clear that some real-world association is being represented. Given this, we will look for associations between organization information and funding information in the source to see if one of these can be used to associate data in the target. For our example, `organizations.code` corresponds to `companies.name`, while `fundings.fid` corresponds to `grants.gid`. Hence, our algorithm will search for associations between these source elements within the source schema. In our example, the referential constraint f_1 indicates that each grant is associated with a company, thus this constraint can be used to associate each company with a set of grants. In general, there may be many ways to associate elements within a schema. Our algorithm will use logical inference to find all associations represented by referential constraints and a schema's relational and nesting structure.

For our example, we have only one way of associating company names and grant gids in the source, so we will use this association to associate fundings with organizations in the target. A mapping reflecting this association is represented by the following formula.

$$\forall \underline{n}, d, y, \underline{g}, a, s, m\ \texttt{companies}(\underline{n}, d, y), \texttt{grants}(\underline{g}, \underline{n}, a, s, m) \rightarrow$$
$$\exists y', \mathrm{F}, f\ \texttt{organizations}(\underline{n}, y', \mathrm{F}),\ \mathrm{F}(\underline{g}, f) \qquad (2)$$

The variable F in formula (2) does not represent an atomic value, but rather a set identifier, and is also used as a term in the formula. This variable represents the set of `fundings` that an `organizations` tuple has.

Notice that Mapping (2) specifies what must be true of the target data, given that a certain pattern holds in the source data. In this example, it says that if a grant joins with a company in the source, then there must be an organization in the target with the name of the company as its code, and with a fundings record nested inside of the organization record that has the grant's gid as its `fundings.fid`. No other constraints are placed on what is in this set. So the mapping is specifying that the association between grants and companies should be preserved in the target.

Now let us consider the correspondence v_3. Considered in isolation, this correspondence could be represented by the following mapping.

$$\forall g, r, \underline{a}, s, m\ \texttt{grants}(g, r, \underline{a}, s, m) \ \rightarrow\ \exists f, p\ \texttt{finances}(f, \underline{a}, p) \tag{3}$$

However, this mapping does not recognize that grant amounts are associated with specific grant gids (in the source) and that `fundings.fid` and `finances.budget` are associated in the target (through the referential constraint f_4). If these two associations represent the same semantic association, then a better mapping can be constructed by using the source and target associations.

$$\forall \underline{n}, d, y, \underline{g}, \underline{a}, s, m\ \texttt{companies}(\underline{n}, d, y), \texttt{grants}(\underline{g}, \underline{n}, \underline{a}, s, m) \ \rightarrow$$

$$\exists y', \mathrm{F}, f, p\ \texttt{organizations}(\underline{n}, y', \mathrm{F}),\ \mathrm{F}(\underline{g}, f), \texttt{finances}(f, \underline{a}, p), \tag{4}$$

Notice that a company and grant tuple that join in the source will create three tuples in the target: an `organizations` tuple, a `fundings` tuple (which is nested inside the `organizations` tuple), and a `finances` tuple. The mapping specifies that the `fundings` and `finances` tuples must share the same value (f) in their `finId` attributes. It does not, however, specify what this value must be (that is, the variable f is existentially quantified and is not bound to source data).

Now to complete our example, let us consider the final correspondence v_4. In the target, a phone is associated with a budget because they are represented in the same relation `finances`. In the source, there are two ways to associate a `grants.amount` (the source for `finances.budget`) and a `contacts.phone` (the source for `finances.phone`). These are represented by the two referential constraints f_2 (which associates a grant with its supervisor's phone) and f_3 (which associates a grant with its manager's phone).

It is not clear which, if either, of these associations should be used to create `finances` tuples. Clio will create two mappings, one using f_2 (Mapping (5)) which uses a join on `supervisor`, and one using f_3 (Mapping (6)) which uses a join on `manager`. To help a user decide which mapping to use, Clio provides a data viewer which allows users to see (and compare) sample target data created by each mapping [51].

$$\forall \underline{n}, d, y, \underline{g}, \underline{a}, s, m, e, \underline{p}\ \texttt{companies}(\underline{n}, d, y), \texttt{grants}(\underline{g}, \underline{n}, \underline{a}, s, m), \texttt{contacts}(s, e, \underline{p})$$

$$\rightarrow\ \exists y', \mathrm{F}, f\ \texttt{organizations}(\underline{n}, y', \mathrm{F}),\ \mathrm{F}(\underline{g}, f), \texttt{finances}(f, \underline{a}, \underline{p}) \tag{5}$$

$$\forall \underline{n}, d, y, \underline{g}, \underline{a}, s, m, e, \underline{p}\ \texttt{companies}(\underline{n}, d, y), \texttt{grants}(\underline{g}, \underline{n}, \underline{a}, s, m), \texttt{contacts}(m, e, \underline{p})$$

$$\rightarrow\ \exists y', \mathrm{F}, f\ \texttt{organizations}(\underline{n}, y', \mathrm{F}),\ \mathrm{F}(\underline{g}, f), \texttt{finances}(f, \underline{a}, \underline{p}) \tag{6}$$

We have illustrated a few of the issues involved with generating schema mappings. We now highlight some of the features of the Clio mapping algorithm.

Mapping Formalism. As our example illustrated, we use source-to-target tgds, a generalization of the relational tgds of Fagin et al. [19] to the nested relational model, to represent schema mappings. Our mappings are a form of what has been called sound GLAV (global-and-local-as-view) mappings [33]. In general, a GLAV mapping asserts a relationship between a query over the source and a query over the target. We use sound mappings, where the relationship between queries is a containment relationship (the result of the source query is contained in the target query) as is common in data integration. Such mappings do not restrict what data can be in the target; hence, we have the freedom to map multiple sources into a single target.

Mapping Generation. Clio exploits the schema and its constraints to generate a set of alternative mappings. Our approach uses the chase technique [36] to generate all possible associations between source elements (and all possible associations between target elements). The mappings are formed using these associations or other associations given by a user.

Multiple Mappings. As we create mappings, each query may omit information that the user may have wanted included. Consider Mapping (2). This mapping takes grants that have an associated company and creates target data from this information. Notice, however, that companies that have no grants would not be included in the mapping. We may want to include all companies, not just companies with grants, in the target. To allow for this, Clio generates a set of mappings that would map all source data (in this example, both companies with and without grants). A user can then choose among these mappings. If she only wishes to map a subset of the source data, she can do so by selecting an appropriate subset of the mappings Clio generates.

2.2 Query Generation for Data Exchange

For data exchange, Clio can generate code that, given a source instance, will produce an instance of the target schema that satisfies the mapping and that represents the source data as accurately as possible. In general, given a source instance there may be many target instances that satisfy the mapping (or many *solutions* in data exchange parlance [19]). Hence, to perform data exchange, we must choose a single "best" target instance, i.e., a single *solution* to materialize.

Let us assume, for example, that the instance of the source schema of Figure 1 is the one illustrated in Figure 2, and that a user has indicated that Mapping (5) is correct and would like to create a target instance. For each `companies` tuple that joins with `grants` and `contacts`, the mapping indicates that there should be an `organizations` tuple containing as organization `code` the company `name`. Furthermore, there must also be a nested `fundings` element containing the `gid` of the grant from the source. Finally, a `finances` element must also exist with the same value for its `finId` as the value of `finId` in this `fundings` record. Moreover, the `finances` element must contain the grant amount in the `budget` element and the supervisor phone number as `phone`. An instance that satisfies

Companies

name	address	year
MS	Redmond, SA	1975
AT&T	Dallas, TX	1983
IBM	Armonk, NY	1911

Grants

gid	recipient	amount	supervisor	manager
g1	MS	1M	Rice	Gates
g2	MS	2M	Bama	Gates
g4	AT&T	3M	Greer	Dorman

Contacts

cid	email	phone
Rice	rice@microsoft	7062838
Gates	gates@microsoft	7069273
Bama	bama@microsoft	7066252
Greer	rxga@att	3607270
Dorman	dorman@att	3600102

Fig. 2. An instance for the source schema in Figure 1

the mapping, i.e., a solution, can be seen in Figure 3. In this solution, `fundings` tuple `g1` can correctly be joined with the first `finances` tuple to associate it with its correct budget (`1M`). However, in Figure 3 all `fundings` tuples are associated with all `finances` tuples, which was not true in the source. In data exchange, we would like the target instance to represent *only* the data associations in the source. Clearly the instance of Figure 3 does not fullfill that desire. Furthermore, the last tuple in the Finances table does not correspond to any source data, yet its inclusion in the instance does not violate Mapping (5). So while this instance is a solution, it is not minimal.

Fagin et al. [19] proposed the use of universal solutions for data exchange. A *universal solution* is an instance of the target schema that contains no more and no less than what the mapping specification requires. A universal solution for Mapping (5) is given in Figure 4. Note that the values of the `finId` attribute have been created to associate each `fundings` with *all* and *only* the `finances` tuples that contain the correct budget amount. The instance is also minimal in that it does not contain any tuples that were not required to be in the target. To compute a universal solution, Fagin et al. [19] present an algorithm based on the chase [36]. However, in Clio, we use queries to perform the data exchange. Here we present algorithms that, given a schema mapping, generate a set of executable queries to perform the desired data exchange. The queries can be in SQL, XQuery or XSLT depending on the desired runtime environment. In the case of a pure relational setting (relational source and target schemas), these queries generate a universal solution. In our algorithm, we make use of Skolem functions (one-to-one functions) that generate values based on a set of source values. We will discuss later how to determine the correct arguments for these Skolem functions and when one Skolem function should be reused in different schema elements. For example, in Section 5, we show why the Skolem function for `finId` needs to depend on four source attributes (`name`, `gid`, `amount`, and `phone`).

In the case of a nested target schema, Clio applies additional grouping and nesting to produce a target instance that is in PNF (Partitioned Normal

Organizations

code	**year**
MS	

Fundings

fid	**finId**
g1	10
g2	10

code	**year**
AT&T	

Fundings

fid	**finId**
g4	10

Finances

finId	**budget**	**phone**
10	1M	7062838
10	2M	7069273
10	3M	3607270
10	5M	2609479

Fig. 3. A non-universal solution target instance for Mapping (5)

Form) [1]. This is done to minimize the redundancy in the target instance. Consider the universal solution given in Figure 4 which contains a different `organizations` tuple with a singleton `fundings` set for each `companies` and `grants` pair in the source, even if multiple pairs share the same company name. Clio avoids this redundancy, by producing a single `organizations` tuple for each source `name` and grouping all the `fundings` that belong to the same organization together under one single organization element (Figure 5). As shown in the figure, we use Skolem functions to represent set identifiers for each `fundings` set. Our algorithms determine the correct arguments for these Skolem functions to achieve PNF grouping. In more recent work which will not be covered in this chapter [24], we have considered how to declaratively represent PNF grouping semantics in the mapping specification along with other types of grouping. In this chapter, we will assume PNF is the desired semantics, and we present our solutions for generating PNF target instances.

There are two main innovations in our data exchange algorithm. The first is a new technique for generating Skolem terms to represent existential values and for achieving grouping in the target instance. Second, our algorithm can identify and merge data that are generated by different mappings, but represent the same target entities. Assume, for instance, that the user wanted to populate the target with all the companies, independently of whether they have funding or not. Mapping (5) can generate only companies with grants (i.e., funding), due to the join it performs on the source data. Mapping (1) on the other hand, generates all the companies, but without their potential funding. The desired target instance can be achieved by using both mappings (5) and (1). The resulting instance would be the same as Figure 5 but with an additional `organizations` element for IBM having an empty `fundings` subelement. The MS and AT&T tuples would not be impacted, even though they are produced by both mappings.

Organizations

code	year
MS	

Fundings

fid	finId
g1	Sk_2(MS,g1,1M,7062838)

code	year
MS	

Fundings

fid	finId
g2	Sk_2(MS,g2,2M,7066252)

code	year
AT&T	

Fundings

fid	finId
g4	Sk_2(AT&T,g4,3M,3607270)

Finances

finId	budget	phone
Sk_2(MS,g1,1M,7062838)	1M	7062838
Sk_2(MS,g2,2M,7066252)	2M	7069273
Sk_2(AT&T,g4,3M,3607270)	3M	3607270

Fig. 4. A universal solution target instance for Mapping (5)

3 Mapping Language and Schema Constraints

Schemas and Types. We use a nested relational model to model both relational and XML Schemas. In general, a *schema* is a sequence of labels (called roots), each with an associated *type* τ, defined by the grammar:

$$\tau ::= \mathsf{String} \mid \mathsf{Integer} \mid \mathsf{Set\ of}\ \tau \mid \mathsf{Rcd}[\,l_1 : \tau_1, \ldots, l_n : \tau_n] \mid \mathsf{Choice}[\,l_1 : \tau_1, \ldots, l_n : \tau_n]$$

Types Integer and String are called atomic types, Set of is a set type, and Rcd and Choice are complex types. With respect to XML Schema, we use Set of to model repeatable elements (or repeatable groups of elements), while Rcd and Choice are used to represent the "all" and "choice" *model-groups*. For each set type Set of τ, τ must be an atomic (String or Integer) type or a Rcd type. We do not consider order, that is, Set of represents unordered sets. "Sequence" model-groups of XML Schema are also represented as (unordered) Rcd types.

Instances. An instance of a schema associates each schema root l of type τ with a value v of type τ. For the atomic types, the allowed values are the expected ones (i.e., strings, integers). A value of type $\mathsf{Rcd}[l_1 : \tau_1, \ldots, l_n : \tau_n]$ is an unordered tuple of pairs $[l_1 : v_1, \ldots, l_n : v_n]$ where v_i is a value of type τ_i with $1 \leq i \leq n$. A value of type $\mathsf{Choice}[l_1 : \tau_1, \ldots, l_n : \tau_n]$ on the other hand, is a pair $\langle l_i : v_i \rangle$ where v_i is a value of type τ_i with $1 \leq i \leq n$. With respect to XML, the labels $l_1, \ldots, l_n$ model element names or attribute names, while the values $v_1, \ldots, v_n$ represent the associated contents or value. In our model, we do not distinguish between XML elements and attributes.

Organizations $Sk_0()$

code	**year**
MS	

Fundings $Sk_1(MS)$

fid	**finId**
g1	Sk_2(MS,g1,1M,7062838)
g2	Sk_2(MS,g2,2M,7066252)

code	**year**
AT&T	

Fundings $Sk_1(AT\&T)$

fid	**finId**
g4	Sk_2(AT&T,g4,3M,3607270)

Finances *Sk3()*

finId	**budget**	**phone**
Sk_2(MS,g1,1M,7062838)	1M	7062838
Sk_2(MS,g2,2M,7066252)	2M	7069273
Sk_2(AT&T,g4,3M,3607270)	3M	3607270

Fig. 5. A target instances for Mapping (5) in PNF

A value of type Set of τ is actually an identifier (*set ID*). This identifier is associated to an unordered set $\{v_1, v_2, \ldots, v_n\}$ of values, each being of type τ.[1] This id-based representation of sets faithfully captures the graph-based data models of XML. In a constraint or expression in general, we shall always interpret the equality of two expressions of set type to mean the equality of their set ids.

Mapping Language. Our mapping language is based on the source-to-target tgds [19], extended to support nested relations. When restricted to the relational model, our mapping formulas coincide with source-to-target tgds. However, we permit variables occurring inside atoms in a tgd to represent relations as well as atomic values to allow the representation of nested sets.

In this paper, as in other Clio work [43,49,52], we will use a more verbose form of the mapping language to make the algorithms and ideas easier to follow. Let an expression be defined by the grammar $e ::= S \mid x \mid e.l$, where x is a variable, S is a schema root, l is a label, and $e.l$ is record projection. Then a *mapping* is a statement (constraint) of the form:

$$\begin{array}{rll} \mathcal{M} ::= & \underline{\text{foreach}} & x_1 \ \underline{\text{in}}\ g_1, \ldots, x_n \ \underline{\text{in}}\ g_n \\ & \quad \underline{\text{where}} & B_1 \\ & \underline{\text{exists}} & y_1 \ \underline{\text{in}}\ g'_1, \ldots, y_m \ \underline{\text{in}}\ g'_m \\ & \quad \underline{\text{where}} & B_2 \\ & \underline{\text{with}} & e_1 = e'_1 \ \underline{\text{and}}\ \ldots \underline{\text{and}}\ e_k = e'_k \end{array}$$

where each x_i in g_i (y_j in g'_j) is called a *generator* and each g_i (g'_j) is one of the following two cases:

1. An expression e with type Set of τ; in this case, the variable x_i (y_j) binds to individual elements of the set e.

[1] The reason will become clear when we talk about how data generated by different mappings is merged to form one single instance of the target schema.

2. A choice selection $e \to l$ (where e is an expression with a type Choice $[\ldots, l : \tau, \ldots]$) representing the selection of attribute l of the expression e; in this case, the variable x_i (y_j) will bind to the element of type τ under the choice l of e.

The mapping is well-formed if the variable (if any) used in g_i (g'_j) is defined by a previous generator within the same clause. Every schema root used in the foreach must be a root of the source schema. Similarly, every schema root used in the exists clause must be a target schema root. The two where clauses (B_1 and B_2) are conjunctions of equalities between expressions over $x_1, \ldots, x_n$, or $y_1, \ldots, y_m$, respectively. They can also include equalities with constants (i.e., selections). Finally, each equality $e_i = e'_i$ in the with clause involves a source expression e_i (over $x_1, \ldots, x_n$) and a target expression e'_i (over $y_1, \ldots, y_m$), with the requirement that e_i and e'_i are of the same atomic type.

For simplicity of presentation, we shall ignore for the remainder of this paper the Choice types and their associated expressions. We note, however, that these are an important part of XML Schema and XML mappings, and they are fully supported in the Clio system.

Example 1. Recall the schemas in Figure 1. The following mapping is an interpretation of the correspondences v_1, v_2, v_3, given the structure of the schemas and the constraints. It is equivalent to the tgd for Mapping (4) in Section 2.

```
foreach  c in companies, g in grants
   where  c.name = g.recipient
exists   o in organizations, f in fundings, f′ in finances
   where  f.finId = f′.finId
with     c.name = o.code and  g.gid = f.fid and  g.amount = f′.budget
```

Each mapping is, essentially, a source-to-target constraint of the form $Q^S \rightsquigarrow Q^T$, where Q^S (the foreach clause and its associated where clause) is a query over the source and Q^T (the exists clause and its associated where clause) is a query over the target. The mapping specifies a containment assertion: for each tuple returned by Q^S, there must exist a corresponding tuple in Q^T. The with clause makes explicit how the values selected by Q^S relate to the values in Q^T.

Schema Constraints (NRIs). Before defining the schema constraints that we use in our mapping generation algorithm, we need to introduce *primary paths* and *relative paths.*

Definition 1. *A* primary path *with respect to a schema root R is a well-formed sequence P of generators x_1 in $g_1, \ldots, x_n$ in g_n where the first generator g_1 is an expression that depends only on R and where each generator g_i with $i \geq 2$ is an expression that depends only on the variable x_{i-1}. Given a variable x of type τ, a* relative path *with respect to x is a well-formed sequence $P(x)$ of generators x_1 in $g_1, \ldots, x_n$ in g_n where the first generator g_1 is an expression that depends only on x and where each generator g_i with $i \geq 2$ is an expression that depends only on x_{i-1}.*

The following are examples of primary paths and relative paths:

$$
\begin{array}{rc}
P_1^S: & c \ \underline{\mathsf{in}} \ \texttt{companies} \\
P_1^T: & o \ \underline{\mathsf{in}} \ \texttt{organizations} \\
P_2^T: & o \ \underline{\mathsf{in}} \ \texttt{organizations}, f \ \underline{\mathsf{in}} \ o.\texttt{fundings} \\
P_r^T: & f \ \underline{\mathsf{in}} \ o.\texttt{fundings}
\end{array}
$$

The first two paths, P_1^S and P_1^T, are primary paths corresponding to top-level tables in the two schemas in our example. The third one is a primary path corresponding to the nested set of `fundings` under the top-level `organizations`. Finally, the fourth one is a relative path with respect to $o : \tau$ where τ is the record type under `organizations`.

Primary paths will play an important role in the algorithm for mapping generation, as they will be the building blocks for larger associations between data elements. Primary paths, together with relative paths, are also useful in definining the schema constraints that we support in our nested relational model. Schema constraints use a similar formalism as mappings, but they are defined within a single schema. Hence, variables in both the $\underline{\mathsf{foreach}}$ and the $\underline{\mathsf{exists}}$ clauses are defined on the same schema.

Definition 2. *A* nested referential integrity constraint (NRI) *on a schema is an expression of the form*

$$\underline{\mathsf{foreach}} \ P_1 \ \underline{\mathsf{exists}} \ P_2 \ \underline{\mathsf{where}} \ B,$$

with the following requirements: (1) P_1 is a primary path of the schema, (2) P_2 is either a primary path of the schema or a relative path with respect to one of the variables of P_1, and (3) B is a conjunction of equalities of the form $e_1 = e_2$ where e_1 is an expression depending on one of the variables of P_1 and e_2 is an expression depending on one of the variables of P_2.

As an example, the following NRI captures the constraint f_4 informally described in Section 2:

$$
\begin{array}{l}
\underline{\mathsf{foreach}} \ \ o \ \underline{\mathsf{in}} \ \texttt{organizations}, f \ \underline{\mathsf{in}} \ o.\texttt{fundings} \\
\underline{\mathsf{exists}} \ \ \ f' \ \underline{\mathsf{in}} \ \texttt{finances} \\
\quad \underline{\mathsf{where}} \ f.\texttt{finId} = f'.\texttt{finId}
\end{array}
$$

Note that NRIs, which capture relational foreign key and referential constraints as well as XML keyref constraints, are a special case of (intra-schema) tgds and moreover such that the $\underline{\mathsf{foreach}}$ and the $\underline{\mathsf{exists}}$ clauses form paths. In general, the source-to-target tgds expressing our mappings may *join* paths in various ways, by using equalities in the $\underline{\mathsf{where}}$ clause on the source and/or the target side. As an example, the mapping in Example 1 in this section joins the primary path for `companies` with the primary path for `grants` in the source, and joins the primary path for `fundings` with the primary path for `finances` in the target. The next section is focused on deriving such mappings.

4 Schema Mapping

Our mapping generation algorithm makes use of associations between atomic elements within the source and target schemas. To describe our approach, we first present an algorithm for finding natural associations within a schema. We then present an algorithm that given a set of correspondences, and a set of source and target associations, creates a set of schema mappings.

4.1 Associations

We begin by considering how the atomic elements within a schema can be related to each other. We define the notion of *association* to describe a set of associated atomic type schema elements. Formally, an association is a form of query (with no return or select clause) over a single schema; intuitively, all the atomic type elements reachable from the query variables (without using additional generators) are considered to be "associated".

Definition 3 (Association). *An* **association** *is a statement of the form:*

$$\begin{array}{ll}\underline{\text{from}} & x_1 \ \underline{\text{in}}\ g_1, \ldots, x_n \ \underline{\text{in}}\ g_n \\ \underline{\text{where}} & B\end{array}$$

where each x_i in g_i is a generator (as defined above). We require that the variable (if any) used in g_i is defined by a previous generator within the same clause. The where clause (B) is a conjunction of equalities between expressions over $x_1, \ldots, x_n$. This clause may also include equalities with constants (i.e., selection conditions).

An association implicitly defines a relationship among all the atomic elements defined by expressions over the variables $x_1, \ldots, x_n$. As a very simple example, consider the following association:

$$\underline{\text{from}}\ c\ \underline{\text{in}}\ \texttt{contacts}$$

The atomic type elements that are implicitly part of this association are c.`cid`, c.`email` and c.`phone`.

To understand and reason about mappings and rewritings of mappings, we must understand (and be able to represent) relationships between associations. We use renamings (1-1 functions) to express a form of subsumption between associations.

Definition 4. *An association A_1 is* **dominated** *by an association A_2 (denoted as $A_1 \dot{\preceq} A_2$) if there is a renaming function h from the variables of A_1 to the variables of A_2 such that the from and where clauses of $h(A_1)$ are subsets, respectively, of the from and where clauses of A_2. (Here we assume that the where clause of A_2 is closed under transitivity of equality.)*

Definition 5. *The* **union** *of two associations A_1 and A_2 (denoted as $A_1 \sqcup A_2$) is an association whose* from *and* where *clause consist of the contents of the respective clauses of A_1 and A_2 taken together (with an appropriate renaming of variables if they overlap).*

If B is a set of equalities $e{=}e'$, we will abuse notation a bit and use $A \sqcup B$ to denote the association A with the equalities in B appended to its where clause.

Structural Associations. An important way to specify semantically meaningful relationships between schema elements is through the structural organization of the schema elements. Associations that are based on this kind of relationship will be referred to as *structural associations.*

Definition 6 (Structural Association). *A* **structural association** *is an association defined by a primary path P and having no where clause:* from *P.*

Example 2. Figure 6 indicates all the primary paths of the two schemas of Figure 1. There is one structural association for each primary path. Note that where more than one relation is used, the relations are related through nesting. For example, $P_2^{\mathcal{T}}$ represents fundings (which will each necessarily have an associated organization).

$P_1^{\mathcal{S}}$: p in `companies`
$P_2^{\mathcal{S}}$: g in `grants`
$P_3^{\mathcal{S}}$: c in `contacts`

$P_1^{\mathcal{T}}$: o in `organizations`
$P_2^{\mathcal{T}}$: o in `organizations`, f in o.`fundings`
$P_3^{\mathcal{T}}$: f in `finances`

Fig. 6. Source and target primary paths

All primary paths (and therefore all structural associations) in a schema can be computed by a one time traversal over the schema [43].

User Associations. The semantic relationships between atomic elements described by structural associations are relationships that the data administrators have encoded in the schema during schema design. However, there are relationships that can exist between schema elements, that are not encoded in the schema, but can be either explicitly specified by a user, or identified through other means such as examining the queries in a given query workload. Associations of this kind are referred to as *user associations.*

Definition 7. *A* **user association** *is any association specified explicitly by a user or implicitly though user actions.*

Example 3. If the `grants.gid` contains as its first five letters, the name of the company who *gave* a grant, then we may find in the workload queries that perform this join frequently. Hence, we may define the following user association:

$$\underline{\text{from}}\ \ g\ \underline{\text{in}}\ \texttt{grants},\ c\ \underline{\text{in}}\ \texttt{companies}$$
$$\underline{\text{where}}\ \ \text{Substr}(g.\texttt{gid},1,5) = c.\texttt{name}$$

Logical Associations. Apart from the schema structure or a user query, an additional way database designers may specify semantic relationships between schema elements is by using constraints.

Example 4. Every record in `grants` in an instance of the source schema of Figure 1 is related to one or more records in `companies` through the referential constraint f_1 from the `recipient` element to `name`. From that, it is natural to conclude that the elements of a grant are semantically related to the elements of a company. Thus, they can be combined together to form an association. Similarly, we can conclude that the elements of a grant are also related to the elements of `contacts` through both of the referential constraints on `supervisor` and `manager`. Formally, the resulting association is:

$$\begin{aligned}&\underline{\text{from}}\ g\ \underline{\text{in}}\ \ \texttt{grants}\ ,\ c\ \underline{\text{in}}\ \ \texttt{companies},\ s\ \underline{\text{in}}\ \ \texttt{contacts},\ m\ \underline{\text{in}}\ \ \texttt{contacts}\\ &\underline{\text{where}}\ g.\texttt{recipient} = c.\texttt{name}\ \text{and}\ g.\texttt{supervisor} = s.\texttt{cid}\\ &\qquad\quad \text{and}\ g.\texttt{manager} = m.\texttt{cid}\end{aligned}$$

There are no other schema elements that can be added to the association by following the same reasoning (based on constraints). In that sense, the association that has just been formed is maximal.

Associations that are maximal like the one described in the previous example are called *logical associations*. Logical associations play an important role in the current work, since they specify possible join paths, encoded by constraints, through which schema elements in different relations or different primary paths can be related. We shall make use of this information in order to understand whether (and how) two correspondences in a schema mapping scenario should be considered together when forming a mapping.

Logical associations are computed by using the chase [36]. The chase is a classical method that was originally introduced to test the implication of functional dependencies. The chase was introduced for the relational model and later extended [42] so that it can be applied on schemas with nested structures.

The chase consists of a series of *chase steps*. A chase step of association A with an NRI f: $\underline{\text{foreach}}\ X\ \underline{\text{exists}}\ Y\ \underline{\text{with}}\ \ B$, can be applied if, by definition, the association A contains the path X (up to a renaming α of the variables in X) but does not satisfy the constraint, that is, A does not contain the path Y (up to a renaming β of the variables in Y) such that B is satisfied (under the corresponding renamings α and β). The result of the chase step is an association A' with the Y clause and the B conditions (under the respective renamings) added to it. The chase can be used to enumerate logical join paths, based on the set of referential constraints in a schema. We use an extension of a nested chase [42] that can handle choice types and NRIs [48].

Let Σ denote a set of NRIs, and A an association. We denote by $\texttt{Chase}_\Sigma(A)$ the final association that is produced by a sequence of chase steps starting from A as follows. In the case where no constraints in Σ can be applied to A, then $\texttt{Chase}_\Sigma(A)$ is A. If there are constraints that can be applied to A, then $\texttt{Chase}_\Sigma(A)$ is constructed as follows: a first step is applied on A; each subsequent step is applied on the output of the previous one; each step uses an NRI from the set Σ; and the sequence of chase steps continues until no more can be applied using NRIs from Σ. In general, if the constraints are cyclic, the chase process may not terminate. However, if we restrict Σ to certain classes of constraints such as *weakly-acyclic* sets of NRIs, termination is guaranteed. (See also the later discussion in Section 6.) Furthermore, it is known (see [19] for example) that when the chase terminates, then it terminates with a unique result (up to homomorphic equivalence[2]). Thus, the result $\texttt{Chase}_\Sigma(A)$ is independent, logically, of the particular order in which the constraints are applied.

Of particular interest to us is the chase applied to structural associations, which is used to compute logical relationships that exist in the schema. A logical association can then be formally defined as the result of such chase.

Definition 8 (Logical Association). *A* **logical association** *is an association* $\texttt{Chase}_\Sigma(A)$*, where A is a structural association or a user association, and Σ is the set of NRIs defined on the schema.*

Example 5. Consider the structural association defined using P_1^S shown in Figure 6. A chase step with the referential constraint

$$f_1: \underline{\text{foreach}}\ g\ \underline{\text{in}}\ \texttt{grants}\ \underline{\text{exists}}\ c\ \underline{\text{in}}\ \texttt{companies}\ \underline{\text{where}}\ g.\texttt{recipient} = c.\texttt{name}$$

cannot be applied since f_1 uses grants in its <u>foreach</u> clause. A chase step with f_1 can, however, be applied on the association defined by P_2^S, since there is a renaming function (in this case the identity mapping) from the variables in the <u>foreach</u> clause of the constraint to the variables of the association. Applying the chase step will lead to an association that is augmented with the <u>exists</u> and <u>where</u> clauses of the constraint. Thus, the association becomes:

$$\begin{array}{l}\underline{\text{from}}\ g\ \underline{\text{in}}\ \texttt{grants},\ c\ \underline{\text{in}}\ \texttt{companies}\\ \underline{\text{where}}\ g.\texttt{recipient} = c.\texttt{name}\end{array}$$

Performing a subsequent chase step with the referential constraint f_2 creates the association:

$$\begin{array}{l}\underline{\text{from}}\ g\ \underline{\text{in}}\ \texttt{grants}\ ,\ c\ \underline{\text{in}}\ \texttt{companies},\ s\ \underline{\text{in}}\ \texttt{contacts}\\ \underline{\text{where}}\ g.\texttt{recipient} = c.\texttt{name}\ \text{and}\ g.\texttt{supervisor} = s.\texttt{cid}\end{array}$$

[2] In our context, two associations are homomorphic equivalent if there are homomorphisms in both directions; this also implies that the associations have a unique *minimal* form.

A subsequent chase step with constraint f_3 will create the association:

```
from g in grants , c in companies, s in contacts, m in contacts
where g.recipient = c.name and g.supervisor = s.cid
      and g.manager = m.cid
```

No additional chase steps can be applied to this association, since all the constraints are "satisfied" by it. Thus, the last association is maximal, and it is a logical association. Note how in this logical association the relation `contacts` appears twice. This is due to the fact that there are two different join paths through which one can reach `contacts` from `grants`. One is through the referential constraint f_2 and one through f_3.

Figure 7 indicates the logical associations that can be generated using all the structural associations of our two example schemas of Figure 1.

```
A1^S: from c in companies
A2^S: from g in grants , c in companies, s in contacts, m in contacts
      where g.recipient = c.name and g.supervisor = s.cid and g.manager = m.cid
A3^S: from c in contacts

A1^T: from o in organizations
A2^T: from o in organizations, f in o.fundings, i in finances
      where f.finId = i.finId
A3^T: from f in finances
```

Fig. 7. Source and target logical associations

4.2 Mapping Generation

Logical associations provide a way to meaningfully combine correspondences. Intuitively, a set of correspondences whose source elements all occur in the same source logical association, and whose target elements all occur in the same target logical association can be interpreted together. Such a set of correspondences maps a set of related elements in the source to a set of related elements in the target data.

The algorithm for generating schema mappings finds maximal sets of correspondences that can be interpreted together by testing whether the elements they match belong in the same logical association, in the source and in the target schema. As seen in the previous section, logical associations are not necessarily disjoint. For example, $A_1^{\mathcal{S}}$ and $A_2^{\mathcal{S}}$ of Figure 7 both include elements of `companies`, although $A_2^{\mathcal{S}}$ also includes elements of `grants` and `contacts`. Thus, a correspondence can use elements that may occur in several logical associations (in both source and target). Rather than looking at each individual correspondence, the mapping algorithm looks at each pair of source and target logical associations. For each such pair, we can compute a candidate mapping that includes all correspondences that use only elements within these logical associations. (As we shall see, not all candidate mappings are actually generated, since some are subsumed by other mappings).

We begin by formally defining correspondences as mappings. First, we define an intensional notion of a *schema element* that we shall use in the subsequent definitions and algorithm.

Definition 9. *Given a schema, a* schema element *is a pair* $\langle P; e\rangle$ *where* P *is a primary path in the schema and* e *is an expression depending on the last variable of* P*.*

Intuitively, the pair $\langle P; e\rangle$ encodes the navigation pattern needed to "reach" all instances of the schema element of interest. Note that a pair $\langle P; e\rangle$ can be turned into a query $\underline{\textsf{select}}$ e $\underline{\textsf{from}}$ P that actually retrieves all such instances.

For our example, $\langle c\ \underline{\textsf{in}}\ \texttt{companies}; c.\texttt{name}\rangle$ represents the schema element `name` in the `companies` table of our source schema, while $\langle o\ \underline{\textsf{in}}\ \texttt{organizations}, f\ \underline{\textsf{in}}\ o.\texttt{fundings}; f.\texttt{fid}\rangle$ identifies the schema element `fid` under `fundings` of `organizations` in our target schema.

Definition 10 (Correspondence). *A* correspondence *from an element* $\langle P^S; e_S\rangle$ *of a source schema to an element* $\langle P^T; e_T\rangle$ *of a target schema is defined by the mapping:*

$$v ::= \underline{\textsf{foreach}}\ P^S \underline{\textsf{exists}}\ P^T \underline{\textsf{with}}\ e_S = e_T$$

In practice, a correspondence need not be restricted to an exact equality; we may allow the application of a function f to e_S. In such case the $\underline{\textsf{with}}$ clause would be $f(e_S) = e_T$. Clio does not discover such functions, but does provide a library of common type conversion functions that a user may specify on a correspondence. The system also permits the use of user-defined functions. Similarly, we could have a function applied to several source elements to generate a single target element (for example, `concat(fname,lname) = name`). This type of N:1 correspondence could be created by some matchers [11] and used by Clio in mapping generation. To keep the notation for our algorithms simple, we will assume that correspondences are of the form given in Definition 10.

Given a pair of source and target logical associations, we would like to define when (and how) a correspondence v is relevant to this pair. In general, a correspondence may be used in multiple ways with a pair of logical associations. For example, a correspondence for the `phone` element under `contacts` can be used in two ways with the pair of associations $\mathcal{A}_2^S$ and $\mathcal{A}_2^T$ in Figure 7 (to map either the `supervisor` or `manager` phone). In our algorithm, we identify all possible ways of using a correspondence. The following definition formalizes the notion of a single use (which we call *coverage*) of a correspondence by a pair of associations.

Definition 11. *A correspondence* v : $\underline{\textsf{foreach}}$ P^S $\underline{\textsf{exists}}$ P^T $\underline{\textsf{with}}$ $e_S = e_T$ *is* covered *by a pair of associations* $<A^{\mathcal{S}}, A^{\mathcal{T}}>$ *if* $P^S \stackrel{\cdot}{\preceq} A^{\mathcal{S}}$ *(with some renaming function* h*) and* $P^T \stackrel{\cdot}{\preceq} A^{\mathcal{T}}$ *(with some renaming function* h'*). We say in this case that there is a* coverage *of* v *by* $<A^{\mathcal{S}}, A^{\mathcal{T}}>$ *via* $<h, h'>$*. We also say that the* result *of the coverage is the expression* $h(e_S){=}h'(e_T)$*.*

Our algorithm will consider each pair of source and target associations that cover at least one correspondence. For each such pair, we will consider all correspondences that are covered and pick one coverage for each. For each such choice (of coverage), the algorithm will then construct what we call a *Clio mapping.*

Definition 12. *Let $\mathcal{S}$ and $\mathcal{T}$ be a pair of source and target schemas and $\mathcal{C}$ a set of correspondences between them. A* Clio mapping *is a mapping* foreach A^S exists A^T with E*, where A^S and A^T are logical associations in the source and the target schema, respectively, and E is a conjunction of equalities constructed as follows. For each correspondence v in $\mathcal{C}$ that is covered by $<A^{\mathcal{S}}, A^{\mathcal{T}}>$, we choose one coverage of v by $<A^{\mathcal{S}}, A^{\mathcal{T}}>$ via $<h, h'>$ and add the equality $h(e_S) = h'(e_T)$ that is the result of this coverage.*

Note that, in the above definition, only one coverage for each correspondence is considered when constructing a Clio mapping. Different coverages will yield different Clio mappings. The following two examples illustrate the process for two different pairs of associations.

Example 6. Consider the pair of logical associations $<A_1^{\mathcal{S}}, A_1^{\mathcal{T}}>$. We check each of the correspondences in $\{v_1, v_2, v_3, v_4\}$ to see if it is covered by these associations. It is easy to see that the correspondence v_1, which relates the source element $\langle c$ in `companies`; c.`name`$\rangle$ with the target element $\langle o$ in `organizations`; o.`code`$\rangle$, is covered by our pair of associations. There is only one coverage in this case, given by the identity renaming functions. Since no other correspondence is covered, we can form a Clio mapping based on $A_1^{\mathcal{S}}$ and $A_1^{\mathcal{T}}$, with the sole equality that results from v_1 added to the with clause:

```
m_v1: foreach  c in  companies
      exists   o in  organizations
      with     c.name = o.code
```

In this simple example, the mapping happens to be the same as the original correspondence. This is because both of the primary paths of this correspondence are logical associations themselves, and also no other correspondence is covered; hence, mapping m_{v1} is v_1. (Notice that this is the same as Mapping (1) from Section (2), where it was represented in traditional s-t tgd notation).

Example 7. As a more complex example of mapping generation, consider the association pair $<A_2^{\mathcal{S}}, A_2^{\mathcal{T}}>$. Following the same steps, we can determine that the correspondences v_1, v_2 and v_3 are covered, and the following mapping using only these three correspondences can be generated:

```
foreach g in  grants, c in  companies, s in  contacts, m in  contacts
        where  g.recipient = c.name and g.supervisor = s.cid
               and g.manager = m.cid
exists  o in  organizations, f in o.fundings, i in  finances
        where  f.finId = i.finId
with    c.name = o.code and g.gid = f.fid and g.amount = i.budget
```

Consider now our final correspondence:

```
v4: foreach c in contacts
    exists  f in finances
    with    c.phone = f.phone
```

which is also covered by the above two associations. However, since `contacts` appears twice in A_2^S, there are two different renaming functions for the foreach clause of v_4; hence, there are two different coverages. In the first, the variable c (of v_4) maps to s, while in the second, the variable c maps to m. This will lead to the generation of the following two mappings.

M_1:

```
foreach g in grants , c in companies, s in contacts, m in contacts
        where g.recipient = c.name and g.supervisor = s.cid
              and g.manager = m.cid
exists  o in organizations, f in o.fundings, i in finances
        where f.finId = i.finId
with    c.name = o.code and g.gid = f.fid and g.amount = i.budget
        and s.phone = i.phone
```

M_2:

```
foreach g in grants , c in companies, s in contacts, m in contacts
        where g.recipient = c.name and g.supervisor = s.cid
              and g.manager = m.cid
exists  o in organizations, f in o.fundings, i in finances
        where f.finId = i.finId
with    c.name = o.code and g.gid = f.fid and g.amount = i.budget
              and m.phone = i.phone
```

Notice that M_1 and M_2 have two copies of `contacts` in their source query, only one of which is used in the target query. A minimization algorithm (similar to tableau minimization [2]) can be applied to remove the occurrence of `contacts` that is not used in the target. So Mapping M_1 is equivalent to the Mapping (5) of Section 2 which is written in the more common s-t tgd notation, and M_2 is equivalent to Mapping (6) of the same section.

Our mapping generation algorithm is summarized in Algorithm 1. If the source schema has N logical associations and the target schema has M logical associations, there will be $N \times M$ pairs of associations that have to be considered by the algorithm. However, not all of these pairs will generate actual mappings. Some pairs may not cover any correspondences and are discarded. Additionally, some pairs of associations are *subsumed* by other pairs and they are also discarded. More precisely, a pair $<A^S, A^T>$ of associations is *subsumed* by another pair $<X, Y>$ of associations if: (1) $X \dot{\preceq} A^S$ or $Y \dot{\preceq} A^T$, and at least one of these two dominances is strict (i.e., X or Y have strictly smaller number of variables), and (2) the set of correspondences covered by $<X, Y>$ is the same as the set of correspondences covered by $<A^S, A^T>$. Intuitively, all the correspondences that

Algorithm 1: Schema Mapping Generation

Input: A source schema $\mathcal{S}$
A target schema $\mathcal{T}$
A set of correspondences $\mathcal{C}$
Output: The set of all Clio mappings $\mathcal{M}$

```
GenerateMappings($\mathcal{S}$, $\mathcal{T}$, $\mathcal{C}$)
(1)     $\mathcal{M} \leftarrow \emptyset$
(2)     $\mathcal{A}^{\mathcal{S}} \leftarrow$ Logical Associations of $\mathcal{S}$
(3)     $\mathcal{A}^{\mathcal{T}} \leftarrow$ Logical Associations of $\mathcal{T}$
(4)     foreach pair $<A^{\mathcal{S}}, A^{\mathcal{T}}>$ of $\mathcal{A}^{\mathcal{S}} \times \mathcal{A}^{\mathcal{T}}$
(5)         $V \leftarrow \{v \mid v \in \mathcal{V} \wedge v$ is covered by $<A^{\mathcal{S}}, A^{\mathcal{T}}> \}$
(6)         // If no correspondences are covered
(7)         if $V = \emptyset$
(8)             continue;
(9)         // Check if subsumed
(10)        if $\exists <X,Y>$ with $X \dot{\preceq} A^{\mathcal{S}}$ or $Y \dot{\preceq} A^{\mathcal{T}}$, and at least one
            dominance is strict
(11)            $V' \leftarrow \{v \mid v \in \mathcal{C} \wedge v$ covered by $<X, Y> \}$
(12)            if $V' = V$
(13)                continue;
(14)        let $V$ be $\{v_1, \ldots, v_m\}$
(15)        for every $v_i$: let $\Delta_{v_i}$ be $\{<h, h'>\mid v_i$ covered by
            $<A^{\mathcal{S}}, A^{\mathcal{T}}>$ via $<h, h'>\}$
(16)        // For every combination of correspondence coverages
(17)        foreach $(\delta_1, \ldots, \delta_m) \in \Delta_{v_1} \times \ldots \times \Delta_{v_m}$
(18)            $W \leftarrow \emptyset$
(19)            foreach $v_i \in V$
(20)                let $e = e'$ be the equality in $v_i$
(21)                let $\delta_i$ be $<h, h'>$
(22)                add equality $h(e) = h'(e')$ to $W$
(23)            form Clio mapping $M$: foreach $A^{\mathcal{S}}$ exists $A^{\mathcal{T}}$ with $W$
(24)            $\mathcal{M} \leftarrow \mathcal{M} \cup \{M\}$
(25)    return $\mathcal{M}$
```

are covered by $<A^S, A^T>$ are also covered by a "strictly smaller" pair of associations. The heuristic that we apply is to discard the "larger" mapping (based on A^S and A^T) since it does not make use of the "extra" schema components. This heuristic can eliminate a large number of unlikely mappings in practice and is a key ingredient for the efficiency of the algorithm. Additional details regarding the data structures needed to efficiently implement this heuristic are given in Haas et al. [27].

5 Query Generation for Data Exchange

The schema mappings we generate specify how the data of the two schemas relate to each other. For data exchange, a source instance must be restructured and transformed into an instance of the target schema. Fagin et al. [19] have defined the problem as follows:

Definition 13. *Given a source schema $\mathcal{S}$, a target schema $\mathcal{T}$, a set Σ_{st} of source-to-target constraints (i.e., the mappings), and a set Σ_t of target constraints, the* data exchange problem *is the following problem: given a finite source instance $\mathcal{I}$, find a finite target instance $\mathcal{J}$ such that $(\mathcal{I}, \mathcal{J})$ satisfies Σ_{st} and $\mathcal{J}$ satisfies Σ_t. Such an instance $\mathcal{J}$ is called a* solution *to the data exchange problem.*

This section will describe one approach to finding such a *solution*. We will discuss in Section 6 how this solution relates to the *universal solution* of Fagin et al. [19].

Notice that a schema mapping (1) does not specify all target values and (2) does not specify the grouping or nesting semantics for target data. Thus, in generating a solution we will have to address these issues. Furthermore, in generating a target instance, we will typically be using many mappings, as we may have one or more mappings per concept in the schemas. Hence, we will need to merge data produced from multiple mappings.

5.1 Intuition: What Are the Challenges

To begin, we will explore a bit further the challenges we face in building our data exchange queries, building up intuition for the algorithm itself.

Creation of New Values in the Target. Consider the simple mapping scenario of Figure 8(a). The source (`Emps`) and the target (`Emps′`) are sets containing employee records. An employee record in the source has atomic elements `A`, `B` and `C`, while an employee record in the target has elements `A′`, `B′`, `C′` along with an extra atomic element `E′`. For the purpose of this discussion, we choose to use the abstract names `A`, etc., so that we can associate several different semantics with these elements for illustration. In the mapping, the two source elements `A` and `B` are mapped into the target elements `A′` and `B′`, while `C′` and `E′` in the target are left unmapped. Consider the following schema mapping which does not specify any value for `C′` or `E′`.

```
foreach  e   in Emps
exists   e′  in Emps′
with     e′.A′ = e.A  and  e′.B′ = e.B
```

To populate the target we need to decide what values (if any) to use for unmapped elements. A frequent case in practice is one in which an unmapped element does not play a crucial role for the integrity of the target. For example, `A` and `B` could be employee name and salary, while `C` and `E` could be spouse and date of birth, respectively, where neither is used in any schema constraint. Creating a null value for `C′` and `E′` is then sufficient. If the unmapped element is optional in a target XML schema, then we can omit this element in the exchange.

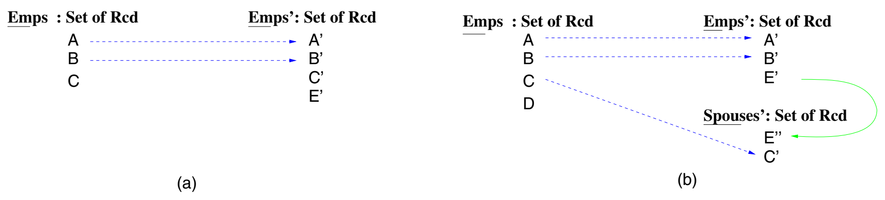


Fig. 8. Creation of new values in the target

Alternatively, the element $\mathtt{E}'$ may be a key in the target relation, e.g., $\mathtt{E}'$ could be employee id. The intention of the mapping would be, in this case, to copy employee data from the source, and assign a new id for each employee in the target. Thus, a non-null (and distinct) value for $\mathtt{E}'$ is needed for the integrity of the target. In general, a target element is **needed** if it is (part of) a key or referential constraint (such as a foreign key) or is both not nullable and not optional.

If $\mathtt{E}'$ is a key, we create a *different* but *unique* value for $\mathtt{E}'$, for *each* combination of the source values for $\mathtt{A}$ and $\mathtt{B}$ using a one-to-one Skolem function. In this example, values for $\mathtt{E}'$ are created using the function $Sk_{\mathtt{E}'}(\mathtt{A},\mathtt{B})$. We can then augment the schema mapping with explicit conditions in the <u>with</u> clause to provide an appropriate value for all unmapped attributes.

$\underline{\mathsf{foreach}}\ \ e\ \ \underline{\mathsf{in}}\ \mathtt{Emps}$
$\underline{\mathsf{exists}}\ \ e'\ \ \underline{\mathsf{in}}\ \mathtt{Emps}'$
$\underline{\mathsf{with}}\ \ e'.A' = e.A\ \underline{\mathsf{and}}\ \ e'.B' = e.B\ \underline{\mathsf{and}}\ \ e'.C' = null\ \underline{\mathsf{and}}\ \ e'.E' = Sk_{\mathtt{E}'}(e.\mathtt{A}, e.\mathtt{B})$

Notice that we choose to make $\mathtt{E}'$ depend only on $\mathtt{A}$ and $\mathtt{B}$, not on the (unmapped) source element $\mathtt{C}$. Thus, even if in the source there may exist two tuples with the same combination for $\mathtt{A}$ and $\mathtt{B}$, but with two different values for $\mathtt{C}$ (e.g., if $\mathtt{C}$ is spouse as above, and an employee has two spouses), in the target there will be only one tuple for the given combination of $\mathtt{A}$ and $\mathtt{B}$ (with one, unknown, spouse). Thus, the semantics of the target is given solely by the values of the source elements that are *mapped*. Of course, a new correspondence from $\mathtt{C}$ to $\mathtt{C}'$ will change the mapping: an employee with two spouses will appear twice in the target and the value for $\mathtt{E}'$ will be $Sk_{\mathtt{E}'}(\mathtt{A},\mathtt{B},\mathtt{C})$.

Now consider an unmapped target element that is used in a *referential constraint*. In Figure 8(b), the (mapped) target element $\mathtt{C}'$ is stored in a different location (the set $\mathtt{Spouses}$) than that of elements $\mathtt{A}'$ and $\mathtt{B}'$. However, the association between $\mathtt{A}',\mathtt{B}'$ values and $\mathtt{C}'$ values is meant to be preserved by a referential constraint ($\mathtt{E}'$ plays the role of a reference in this case). The schema mapping created by Clio is the following.

$\underline{\mathsf{foreach}}\ \ e\ \ \underline{\mathsf{in}}\ \mathsf{Emps}$
$\underline{\mathsf{exists}}\ \ e'\ \ \underline{\mathsf{in}}\ \mathsf{Emps}',\ s'\ \ \underline{\mathsf{in}}\ \mathsf{Spouses}'$
$\quad\underline{\mathsf{where}}\ \ e'.\mathsf{E}' = s'.\mathsf{E}''$
$\underline{\mathsf{with}}\ \ e'.\mathtt{A}' = e.\mathtt{A}\ \underline{\mathsf{and}}\ \ e'.\mathsf{B}' = e.\mathsf{B}\ \underline{\mathsf{and}}\ \ s'.\mathsf{C}' = e.\mathsf{C}$

Note that this mapping does not give a value for the required element $\texttt{Emps}'.\texttt{E}'$ or $\texttt{Spouses}'.\texttt{E}''$. We can provide values for these two unmapped elements using a Skolem function $Sk_{\texttt{E}'}(\texttt{A},\texttt{B},\texttt{C})$ to create values for $\texttt{E}'$ and $\texttt{E}''$.

```
foreach  $e$  in Emps
exists   $e'$  in $\texttt{Emps}'$, $s'$  in $\texttt{Spouses}'$
    where  $e'.\texttt{E}' = s'.\texttt{E}''$
with     $e'.\texttt{A}' = e.\texttt{A}$ and $e'.\texttt{B}' = e.\texttt{B}$ and $s'.\texttt{C}' = e.\texttt{C}$ and $e'.\texttt{E}' = Sk_{\texttt{E}'}(e.\texttt{A}, e.\texttt{B}, e.\texttt{C})$
```

In the above example, the same Skolem function will populate both E' and E'', since E' and E'' are required to be equal by the where clause of the mapping. In general, however, different target attributes will use different Skolem functions.

Note also that if duplicate $[a, b, c]$ triples occur in the source (perhaps with different D values, where D is some other attribute) only one element is generated in each of $\texttt{Emps}'$ and $\texttt{Spouses}'$. Thus, we eliminate duplicates in the target based on the *mapped* source data.

Grouping of Nested Elements. Consider now Figure 9(a), in which the target schema contains two levels of nesting: elements $\texttt{A}'$ and $\texttt{C}'$ are at the top level, while there are multiple $\texttt{B}'$ elements ($\texttt{Bs}'$ is of set type). Elements A, B, and C of the source Emps are mapped, via correspondences, into the respective elements of the target $\texttt{Emps}'$. The mapping that Clio generates:

```
foreach  $e$  in Emps
exists   $e'$  in $\texttt{Emps}'$, $b'$  in  $e'.Bs'$
with     $e'.\texttt{A}' = e.\texttt{A}$  and  $b'.\texttt{B}' = e.\texttt{B}$  and  $e'.\texttt{C}' = e.\texttt{C}$
```

requires that all $(\texttt{A},\texttt{B},\texttt{C})$ values found in the source appear in the target. In addition, a natural interpretation of the target schema is that all B values sharing the same A and C be grouped together in one set. For illustration, if A, B, and C are employee, child, and spouse names, respectively, Clio will choose to group all children with the same employee and spouse in a single set. Note that this behavior is *not* part of the mapping specification itself. (A solution of the above mapping can be obtained by creating a target tuple with a singleton Bs'-set for each source tuple.) However, for data exchange, we choose to add this grouping

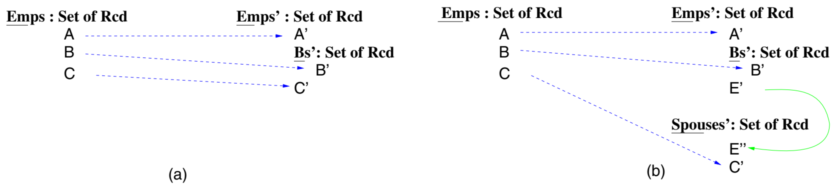


Fig. 9. Grouping of elements in the target

semantics that is implicitly specified by the target schema, and produce a target instance that is in Partitioned Normal Form (PNF) [1].

PNF: *In any target nested relation, there cannot exist two distinct tuples that coincide on all the atomic elements but have different set-valued elements.*

To achieve this grouping behavior, we use Skolemization as well. If a target element has a set type, then its identifier (recall that each set is identified in the data model by a set id) is created via a Skolem function. This function *does not* depend on any of the atomic elements that are mapped under (in terms of nesting) the respective set type, in the schema hierarchy. Instead it depends on all the atomic elements at the same level or above (up to the root of the schema). The same Skolem function (for a given set type of the target schema) is used across all mappings. Intuitively, we perform a deep union of all data in the target independently of their source. For the example of Figure 9(a), we modify the schema mapping with a Skolem function for Bs.

```
foreach  $e$ in Emps
exists   $e'$ in Emps$'$, $b'$ in $e'.Bs'$
with     $e'.\mathtt{A}' = e.\mathtt{A}$ and $b'.\mathtt{B}' = e.\mathtt{B}$ and $e'.\mathtt{C}' = e.\mathtt{C}$ and $e'.Bs' = Sk_{Bs'}(e.\mathtt{A}, e.\mathtt{C})$
```

The meaning of the above rule is the following: for each (a, b, c) triple of values from the source, create a record in Emps$'$, with the appropriate A$'$ and C$'$ attributes, and also a Bs$'$ attribute, the value of which is the set id $Sk_{\mathtt{Bs}'}(a, c)$. Thus, the Skolem function $Sk_{\mathtt{Bs}'}$ is used here to create a set type element. Also, we create an element B$'$, with value b, under $Sk_{\mathtt{Bs}'}(a, c)$. Another tuple (a, b', c) will lead to the value b' being nested in the same set as b.

Hence, we achieve desired grouping of B$'$ elements for fixed A$'$ and C$'$ values.

Value Creation Interacts with Grouping. To create a nested target instance, we will need to consider referential constraints together with our desired PNF grouping. We again explain our technique with an example. Consider Figure 9(b), where the elements A$'$ and C$'$ are stored in separate target sets. The association between A$'$ (e.g., employee name) and C$'$ (e.g., spouse name) is preserved via the foreign key E$'$ (e.g., spouse id). Thus, E$'$ is a required element and must be created. However, in this case, it is rather intuitive that the value of E$'$ should not depend on the value of B$'$, but only on the A$'$ and C$'$ value. This, combined with the PNF requirement, means that all the B$'$ (child) values are grouped together if the employee and spouse names are the same. We achieve therefore the same effect that the mapping of Figure 9(a) achieves. In contrast, if E$'$ were to be assigned a different value for different B$'$ values, then each child will end up in its own singleton set. For data exchange, we choose the first alternative, because we believe that this is the more practical interpretation. Thus, we adjust the earlier Skolemization scheme for atomic type elements as follows.

The function used for creation of an atomic element E *does not depend on any of the atomic elements that occur at a lower level of nesting in the target schema.*

For the example of Figure 9(b) we create the rule:

```
foreach  $e$  in Emps
exists   $e'$  in Emps$'$, $b'$  in  $e'.Bs'$, $s'$  in Spouses$'$
   where  $e'.\mathtt{E}' = s'.\mathtt{E}''$
with     $e'.\mathtt{A}' = e.\mathtt{A}$  and  $b'.\mathtt{B}' = e.\mathtt{B}$  and  $s'.\mathtt{C}' = e.\mathtt{C}$  and
              $e'.\mathtt{E}' = Sk_{\mathtt{E}'}(e.\mathtt{A}, e.\mathtt{C})$  and  $e'.Bs' = Sk_{Bs'}(e.\mathtt{A}, Sk_{\mathtt{E}'}(e.\mathtt{A}, e.\mathtt{C}))$
```

Note that in this case, the Skolem function for Bs$'$ is (as before) a function that depends on all the atomic type elements that are at the same level or above it (in the schema hierarchy). Thus, it is a function of the values that are assigned to A$'$ and E$'$ (the latter being a Skolem function itself, in this case).

As an extreme but instructive case, suppose that in Figure 9(b) we remove the correspondences that involve A$'$ and C$'$, but keep the one that involves B$'$. If the target elements A$'$ and C$'$ are not optional, then they will be created by (unique) Skolem functions with no arguments. This is the same as saying that they will each be assigned a single (distinct) null. Consequently, the two target sets Emps$'$ and Spouses$'$ will each contain a single element: some unknown employee, and some unknown spouse, respectively. In contrast, the nested set Bs$'$ will contain all the B values (all the children listed in Emps). Thus, the top-level part of the schema plays only a structural role: it is *minimally* created in order to satisfy the schema requirements, but the respective values are irrelevant. Later on, as correspondences may be added that involve A$'$ and C$'$, the children will be separated into different sets, depending on the mapped values.

We now describe in some detail the algorithm for generating the data exchange queries. The algorithm is applied to every Clio mapping that has been generated, and works in three steps. First, given a Clio mapping, we construct a graph that represents the key portions of the query to be generated. Second, we annotate that graph to generate the Skolem terms for the query. These two steps are discussed in the next section. Finally, we walk the graph, producing the actual query, as described in Section 5.3.

5.2 The Query Graph

Given a mapping m, a graph is constructed, called the *query graph.* The graph will include a node for each target expression that can be constructed over the variables in the exists clause of the mapping. Source expressions that provide values for the target will also have nodes. Furthermore, the graph will include links between nodes to reflect parent-child relationships and equality predicates, respectively. This structure is then further refined to support Skolem function generation; at that point, it contains all the information needed to generate the query.

In the query generation algorithm, all the variables that appear in the input mapping are assumed to be named consistently as: $x_0, x_1, \ldots$ for the source side, and $y_0, y_1, \ldots$ for the target side. Let us revisit one of our earlier mappings (M_1 in Section 4.2), using the above naming convention:

```
foreach $x_0$ in companies, $x_1$ in grants , $x_2$ in contacts, $x_3$ in contacts
        where $x_0$.name = $x_1$.recipient and $x_1$.supervisor = $x_2$.cid
              and $x_1$.manager = $x_3$.cid
exists  $y_0$ in organizations, $y_1$ in $y_0$.fundings, $y_2$ in finances
        where $y_1$.finId = $y_2$.finId
with    $x_0$.name = $y_0$.code and $x_1$.gid = $y_1$.fid and $x_1$.amount = $y_2$.budget
              and $x_2$.phone = $y_2$.phone
```

The query graph for this mapping is illustrated in Figure 10. The query graph is constructed by adding a node for each variable (and its generator) in the exists clause of the mapping. There are three such nodes in our example query graph. Furthermore, we add nodes for all the atomic type elements that can be reached from the above variables via record projection. For our example, we include nodes for y_0.code, y_0.year, and so on. (If we have multiple levels of records, then there will be several intermediate nodes that have to be introduced.) We then add structural edges (the directed full arcs in the figure) to reflect the obvious relationships between nodes. Moreover, we add *source* nodes for all the source expressions that are actually used in the with clause of the mapping. For our example, we add nodes for x_0.name, x_1.gid, and the other two source expressions. We then attach these source nodes to the target nodes to which they are "equal". This is illustrated via the dotted, directed arcs in the figure (e.g., the arc from y_0.code to x_0.name).

Finally, we use the equalities in the where clause on the target side and add "equality" edges between target nodes. For our example, we add the dotted, undirected, edge connecting the node for y_1.finId and the node for y_2.finId.

Next, we annotate each node in the query graph to facilitate the subsequent generation of Skolem functions. Each node in the graph will be annotated with a set of source expressions. These expressions will represent the *arguments* of a potential Skolem function for that node. However, only the nodes that are

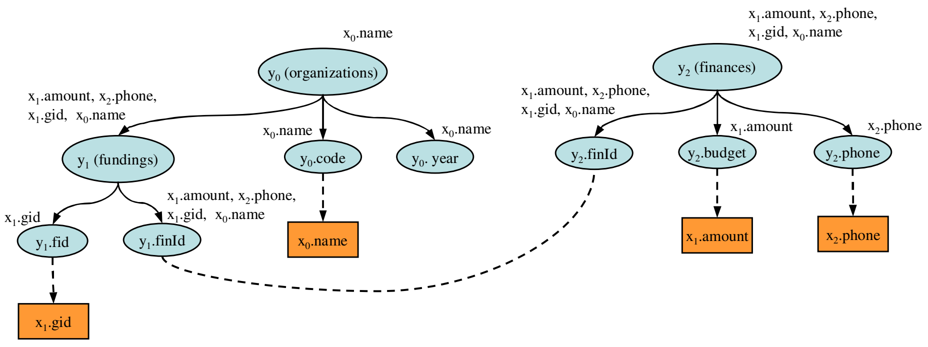


Fig. 10. An annotated query graph

not mapped from the source and that are needed (as discussed in the previous section) will receive an actual Skolem function.

The annotations are computed by a propagation algorithm. First, every target node that is adjacent to a source schema node through an "equality" (dotted) edge, is annotated with the expression of that source schema node and only that. Next, we start propagating these expressions through the rest of the graph according to the following rules.

– **Upward propagation:** Every expression that a node acquires is propagated to its parent node, unless the (aquiring) node is a variable.
– **Downward propagation:** Every expression that a node acquires is propagated to its children if they do not already have it and if they are not equal to any of the source nodes.
– **Eq propagation:** Every expression that a node acquires is propagated to the nodes related to it through equality edges.

The propagation rules are applied repeatedly until no more rules can be applied. Figure 10 illustrates the result of propagation for our example. The resulting annotated graph is now ready for use to generate a data exchange query.

5.3 Generation of Transformation Queries

Once the query graph of a mapping has been generated, it can be converted to an actual query. The language of the query depends on the model of the source and target. When both source and target are relational schemas, the query will be a relational SQL query. When one schema is XML, Clio can generate a data exchange query in XQuery or XSLT. We describe here how to generate a data exchange query in XQuery. The XSLT and SQL approaches are similar, though for SQL, the query will not have nested elements.

The complete XQuery that is generated for our mapping M_1 is shown in Figure 11. An XSLT version of such query can be generated in a similar way, with only syntactic differences.

To begin, the foreach clause of the mapping is converted to a query fragment on the source schema. This is straightforward: Clio simply binds one variable to each term of the mapping and adds the respective conditions in the where clause. For instance, for our mapping M_1, we create the following XQuery fragment:

```
FOR $x0 IN $doc0/companies, $x1 IN $doc0/grants,
    $x2 IN $doc0/contacts, $x3 IN $doc0/contacts,
WHERE $x0.name=$x1.recipient and $x1.supervisor=$x2.cid
and $x1.manager=$x3.cid
```

Let us denote this query fragment by $Q^S_{M_1}$. Note that this is not a complete query since it does not have a result yet. This query fragment will be used (repeatedly) in a larger query that we generate based on the query graph. The algorithm starts at the target schema root in the query graph and performs a depth-first traversal.

– If the node being visited corresponds to a complex type schema element, then a complex element is generated by visiting the children and enclosing their corresponding elements.
– If a node corresponding to an atomic type element is visited, then: (i) if the node is linked to a source node (directly, or via any number of equality edges), then a simple element is generated with the value equal to the expression represented by the source node, or (ii) if the node corresponds to an optional element, nothing is generated, or (iii) if the node corresponds to a nullable schema element, the null value is generated; or finally, (iv) if none of the above applies, a value must be generated for that element via a Skolem function. In this case, we generate a fresh new Skolem function name and add all arguments that annotate the node. We take care so that all the nodes that are "equal" to the current node will receive the same Skolem function name.
– If the node visited is a variable, then a **FOR**-**WHERE**-**RETURN** query is produced, by first taking a "copy" of the source query fragment (e.g., $Q^S_{M_1}$) (where we rename, in a consistent way, all the variables in the query fragment). In addition, we inspect the expressions that annotate the node and compare with its parent variable (if any). For each *common* expression e, we then add an extra equality condition that equates the expression e at the parent query with (the renaming of) e in the current query. This creates a correlated subquery.

  For our example, the subquery that is generated for the node y_1 in our query graph is based on a copy of the query fragment $Q^S_{M_1}$, with an extra equality requiring that (the renaming of) x_0.`name` in the subquery is equal to to x_0.`name` in the parent (e.g., "$x0L1/name/text() = $x0/name/text()").

  The traversal continues and any new elements, generated based on the descendants of the current node (variable), will be placed inside the return clause of the subquery that has just been created.

One issue that we need to address in the actual implementation is the fact that Skolem functions are not first-class citizens in typical query languages, including XQuery. However, this issue can be dealt with in practice by simulating a Skolem function as a string that concatenates the name of the Skolem function with the string representations of the values of the arguments.

The above query generation algorithm, as presented, is not complete in the sense that it may not generate a PNF instance. If we consider each Clio mapping individually, then the generated query does produce a data fragment that is in PNF. However, in the case of multiple mappings, we still need to merge the multiple fragments into one PNF instance.

The merging required by PNF can be achieved either via post-processing, or by generating a more complex, two-phase query as follows. The first query transforms the source data, based on the mappings, into a set of "flat" views that encode (in a relational way) the nested target instance in PNF. Nested sets are encoded in this flat view by using Skolem functions (as described in Section 5.1). These Skolem functions represent the identifiers of sets, and each

```
LET $doc0 := document("input XML file goes here")
RETURN
 <T>
   {distinct-values (
   FOR $x0 IN $doc0/companies, $x1 IN $doc0/grants,
       $x2 IN $doc0/contacts,  $x3 IN $doc0/contacts
   WHERE
       $x0/name/text() = $x1/recipient/text() AND $x1/supervisor/text() = $x2/cid/text() AND
       $x1/manager/text() = $x3/cid/text()
   RETURN
      <organization>
        <code>  {  $x0/name/text() }  </code>
        <year>  {  "Sk3(", $x0/name/text(), ")" }  </year>
        {distinct-values (
        FOR $x0L1 IN $doc0/companies, $x1L1 IN $doc0/grants,
            $x2L1 IN $doc0/contacts,  $x3L1 IN $doc0/contacts
        WHERE
            $x0L1/name/text() = $x1L1/recipient/text() AND $x1L1/supervisor/text() = $x2L1/cid/text()
        AND $x1L1/manager/text() = $x3L1/cid/text() AND $x0L1/name/text() = $x0/name/text()
        RETURN
           <funding>
              <fid> {$x0L1/gid/text()} </fid>
              <finId>{"Sk5(", $x0L1/name/text(), ", ", $x2L1/phone/text(), ", ",
                      $x1L1/amount/text(), ", ", $x1L1/gid/text(), ")"}</finId>
           </funding> ) }
     </organization> ) }
   {distinct-values (
   FOR $x0 IN $doc0/companies, $x1 IN $doc0/grants,
       $x2 IN $doc0/contacts,  $x3 IN $doc0/contacts
   WHERE
       $x0/name/text() = $x1/recipient/text() AND $x1/supervisor/text() = $x2/cid/text() AND
       $x1/manager/text() = $x3/cid/text()
   RETURN
       <financial>
          <finId>{"Sk5(", $x0/name/text(), ", ", $x2/phone/text(), ", ",
                   $x1/amount/text(), ", ", $x1/gid/text(), ")"}</finId>
          <budget>  {  $x1/amount/text() }  </budget>
          <phone>   {  $x2/phone/text() }  </budget>
      </financial> ) }
</T>
```

Fig. 11. The data exchange XQuery for Mapping M_1

tuple records the id of the set where it belongs. The second query can then take this result and merge the data into the right hierarchy, by joining on the set identifier information produced by the first query. The resulting instance is then guaranteed to be in PNF.

The full details for the two-phase query generation can be found in [27] and in [24].

6 Analysis

In this section we discuss the general conditions under which our method for mapping generation is applicable, and we make precise the connection between our query generation algorithm and the data exchange framework of Fagin et al. [19].

6.1 Complexity and Termination of the Chase

The chase is the main process used to construct logical associations. In general, it is known that the chase with general dependencies may not terminate.

The chase that we use is a special case in which the dependencies (the schema constraints) are NRIs. Even in this special case, the chase may not terminate: one can construct simple examples of cyclic inclusion dependencies (even for the relational model) for which the chase does not terminate.

To guarantee termination, we restrict the NRIs within each schema to form a *weakly-acyclic* set. The notion of a weakly-acyclic set of tgds has been studied [19] and, with a slight variation, in [17]; it was shown that weakly-acyclic sets of tgds strictly generalize the previous notions of acyclic dependencies and it was proven that the chase with a weakly-acyclic set of tgds always terminates in polynomially many chase steps. The definition of weak-acyclicity and the argument for termination apply immediately when we consider tgds over the nested relational model. Hence they apply to our NRIs.

While the number of chase steps required to complete the chase is polynomial (in the weak-acyclic case), a chase step itself (of an association with an NRI) can take an exponential amount of time in the worst case. This is due to the fact that the paths in an NRI require matching (on the association) to determine the applicability of the NRI. In general, there could be multiple ways of matching a variable in the path with a variable in an association. Hence, matching a path means exploring an exponential number of assignments of path variables in the worst case. However, the exponential is in the size of the path (i.e., the number of variables), which in practice is often small. Furthermore, in most cases, a variable in a path has only one way of matching due to schema restrictions (e.g., a variable required to be in `companies` can only match with a variable in `companies`). Thus, the exponential worst case rarely materializes.

Another challenge that is often found in XML schemas is type recursion. Type recursion occurs when a complex type is used in its own definition. Type recursion leads to infinitely many structural associations that may result in infinitely

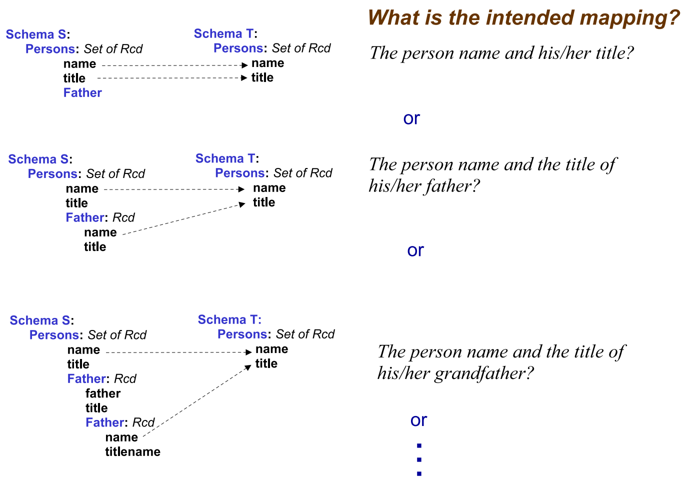


Fig. 12. Unlimited candidate mappings due to recursive types

many possible interpretations of a given set of correspondences. Figure 12 illustrates how type recursion can lead to an infinite number of mappings. In this figure, the element `Father` is of type `Persons`. In order to partially cope with this limitation we allow recursive types but we require the user to specify the recursion level, or we choose a specific constant value as a bound. For example, in the mapping scenario of Figure 12, if we bind the recursion level to one, then only the first two mappings shown in the figure will be generated by the system, and the user will be left to select the one that better describes the intended semantics of the correspondences.

6.2 Characterization of Data Exchange

Consider a data exchange setting, that is, a source schema $\mathcal{S}$, a target schema $\mathcal{T}$, a set of mappings $\mathcal{M}$ from $\mathcal{S}$ to $\mathcal{T}$, a set of target constraints Σ_T, and an instance I of schema $\mathcal{S}$. The problem in data exchange is to find a *solution*, i.e., an instance J of the target schema $\mathcal{T}$ that is consistent with the mappings in $\mathcal{M}$. If we assume that we are in the relational model, this allows us to use some of the existing results in the research literature. In a data exchange setting, there may be more than one solution. Fagin et al. [19] have provided an algebraic specification that selects among all solutions of a data exchange setting, a special solution that is referred to as a *universal solution.* A universal solution intuitively has no more and no less data than required for data exchange, and represents the entire space of possible solutions. A universal solution J_u is a solution for which there is a homomorphism h to every other solution J. Fagin et al. provided an algorithm, based on the chase, for computing a universal solution. That algorithm starts with the source instance I and chases it with the mappings, and also with the target schema constraints. The result of the chase gives a target instance that is a universal solution.

There are two main differences in our approach. First, the process of generating logical associations compiles the schema constraints (NRIs) into the associations and, hence, into the mappings we generate. As a consequence, the resulting set of mappings $\mathcal{M}$ "includes" the effect of the source and target schema constraints. Since in the algorithms presented in this chapter, we do not consider other target constraints (e.g., key constraints or functional dependencies), we reduce the data exchange problem to one based on $\mathcal{M}$ alone.

The other main difference is that we perform data exchange by generating queries that "implement" $\mathcal{M}$. The main property of our query generation algorithm is that for relational schemas the instance J_u generated by our data exchange queries is always a *universal solution.* Indeed, any other solution J will agree with the instance J_u on the elements for which correspondences have been defined (the elements determined by the mapping). For all the other elements, the data exchange queries have generated Skolem values that can be mapped to the corresponding values of instance J. Thus, there is a homomorphism from J_u to any other solution J, which means that J_u is a universal solution.

For nested relational schemas, we additionally took the view that the mapping $\mathcal{M}$ also specifies some implicit grouping conditions on the target data. In particular, we required that the target instance must be in partitioned normal form (PNF): in any set we cannot have two distinct tuples that agree on all the atomic valued components but do not agree on the set-valued elements. Our query generation algorithm has then the property that the data exchange queries we produce will always generate a target instance that is a PNF version of a universal solution. Alternatively, the generated target instance is a PNF solution that is universal with respect to all PNF solutions.

7 Related Work

Declarative schema mapping formalisms have been used to provide formal semantics for data exchange [19], data integration [33], peer data management [28,13,25], pay-as-you-go integration systems [45], and model management operators [8]. A whole area of model management has focused on such issues as mapping composition [35,21,40] and mapping inverse [18,22].

Schema mappings are so important in information integration that many mapping formalisms have been proposed for different tasks. Here we mention only a few. The important role of Skolem functions for merging data has been recognized in a number of approaches [30,41]. HePToX [13] uses a datalog-like language that supports nested data and allows Skolem functions. Extensions to the languages used for schema mapping include nested mappings [24], which permit the declarative specification of correlations between mappings and grouping semantics, including the PNF grouping semantics used in Clio. Clip [44] provides a powerful visual language for specifying mappings between nested schemas. And second-order tgds [21] provide a mapping language that is closed under composition.

Perhaps the only work on mapping discovery that predates Clio is the TranSem system [39] which used matching to select among a pre-specified set of local schema transformations that could help transform one schema into another. Work on mapping discovery has certainly continued. Fuxman et al. [24] consider how to create nested mappings. An et al. [5] consider how to use conceptual schemas to further automate mapping discovery. Yan et al. [51] and Alexe et al. [3] consider how to use data examples to help a user interactively design and refine mappings for relational and nested schemas respectively. Hernández et al. [29] consider the generation of mappings that use data-metadata translations [50]. Also, Bohannon et al. [12] consider the generation of information preserving mappings (based on path mappings).

Analysis of mappings has become an important new topic with work on verification of mappings [14], mapping quality [15,32], and mapping optimization [20], to name just a few.

Many industry tools such as BizTalk Mapper, IBM WebSphere Data Stage TX, and Stylus Studio's XML Mapper support the development (by a programmer) of mappings. Although these tools do not automate the mapping discovery process, they do provide useful programming environments for developing mappings. The Clio algorithms for mapping discovery are used in several IBM products, including IBM Rational Data Architect, and IBM InfoSphere FastTrack. STBenchmark [4] presents a new benchmark for schema mapping systems.

Clio was the first system to generate code (in our case queries) for data exchange. The generation of efficient queries for data exchange is not considered in work like Piazza [28] and HePToX [13] which instead focus on query generation for data integration. More recently, in model management [37,10], query or code generation for data exchange has been considered for embedded dependencies. Hernández et al. [29] generate data exchange queries for richer mappings that include data to metadata conversion. And specialized engines for efficiently executing data exchange queries have been proposed [31].

8 Conclusions

We have presented a retrospective on key contributions of the Clio schema mapping system. These innovations include a new paradigm, in which we view the mapping creation process as one of query discovery. Clio provides a principled algorithm for discovering queries over the source, queries over the target, and a precise specification of their relationship. In Clio, we pioneered the use of schema mapping queries to capture the relationship between data in two heterogeneous schemas. Many uses have been found for such schema mappings, but Clio was the first system to exploit them to perform data exchange between independently created schemas, leading to a new theory of data exchange. In this chapter, we have presented our algorithms for both schema mapping creation via query discovery, and for query generation for data exchange. Our algorithms apply equally in pure relational, pure XML (or any nested relational), and mixed relational and nested contexts.

Clio set out to radically simplify information integration, by providing tools that help users convert data between representations – a core capability for integration. Today, most information integration systems, whether federation engines that do data integration, or ETL engines that enable data exchange, include a suite of tools to help users understand their data and to create mappings, though only a few leverage the power of Clio's algorithms. In the next generation of integration tools, we need to leverage data and semantic metadata more effectively in the integration process, combining data-driven, metadata-driven and schema-driven reasoning. Further, we need to provide users with a higher level of abstraction for the entire integration process, from identification of the data of interest through returning the results. Ideally, users would not have to decide *a priori* whether they wanted data integration or data exchange; instead, the system should understand the user's intentions and construct the integration plan accordingly [26]. These challenges are natural next steps along the trail of increased automation and radical simplification blazed by Clio.

References

1. Abiteboul, S., Bidoit, N.: Non-first Normal Form Relations: An Algebra Allowing Data Restructuring. J. Comput. Syst. Sci. 33, 361–393 (1986)
2. Abiteboul, S., Hull, R., Vianu, V.: Foundations of Databases. Addison-Wesley, Reading (1995)

3. Alexe, B., Chiticariu, L., Miller, R.J., Tan, W.-C.: Muse: Mapping understanding and design by example. In: Proceedings of International Conference on Data Engineering (ICDE), pp. 10–19 (2008)
4. Alexe, B., Tan, W.-C., Velegrakis, Y.: STBenchmark: towards a benchmark for mapping systems. In: Proceedings of the VLDB Endowment, vol. 1(1), pp. 230–244 (2008)
5. An, Y., Borgida, A., Miller, R.J., Mylopoulos, J.: A Semantic Approach to Discovering Schema Mapping Expressions. In: Proceedings of International Conference on Data Engineering (ICDE), pp. 206–215 (2007)
6. Batini, C., Lenzerini, M., Navathe, S.B.: A Comparative Analysis of Methodologies for Database Schema Integration. ACM Computing Surveys 18(4), 323–364 (1986)
7. Beeri, C., Vardi, M.Y.: A proof procedure for data dependencies. J. ACM 31(4), 718–741 (1984)
8. Bernstein, P., Halevy, A., Pottinger, R.: A Vision for Management of Complex Models. SIGMOD Record 29(4), 55–63 (2000)
9. Bernstein, P.A., Haas, L.M.: Information Integration in the Enterprise. Commun. ACM 51(9), 72–79 (2008)
10. Bernstein, P.A., Melnik, S., Mork, P.: Interactive Schema Translation with Instance-Level Mapping. In: Proceedings of the International Conference on Very Large Data Bases (VLDB), pp. 1283–1286 (2005)
11. Bohannon, P., Elnahrawy, E., Fan, W., Flaster, M.: Putting Context into Schema Matching. In: Proceedings of the International Conference on Very Large Data Bases (VLDB), pp. 307–318 (2006)
12. Bohannon, P., Fan, W., Flaster, M., Narayan, P.P.S.: Information Preserving XML Schema Embedding. In: Proceedings of the International Conference on Very Large Data Bases (VLDB), pp. 85–96 (2005)
13. Bonifati, A., Chang, E.Q., Ho, T., Lakshmanan, V.S., Pottinger, R.: HePToX: Marrying XML and Heterogeneity in Your P2P Databases. In: Proceedings of the International Conference on Very Large Data Bases (VLDB), pp. 1267–1270 (2005)
14. Bonifati, A., Mecca, G., Pappalardo, A., Raunich, S., Summa, G.: Schema mapping verification: the spicy way. In: International Conference on Extending Database Technology (EDBT), pp. 85–96 (2008)
15. Bonifati, A., Mecca, G., Pappalardo, A., Raunich, S., Summa, G.: The spicy system: towards a notion of mapping quality. In: ACM SIGMOD Conference, pp. 1289–1294 (2008)
16. Chawathe, S., GarciaMolina, H., Hammer, J., Ireland, K., Papakonstantinou, Y., Ullman, J., Widom, J.: The TSIMMIS Project: Integration of Heterogeneous Information Sources. In: Proc. of the 100th Anniversary Meeting of the Information Processing Society of Japan (IPSJ), Tokyo, Japan, pp. 7–18 (1994)
17. Deutsch, A., Tannen, V.: XML queries and constraints, containment and reformulation. Theoretical Comput. Sci. 336(1), 57–87 (2005)
18. Fagin, R.: Inverting schema mappings. ACM Transactions on Database Systems (TODS) 32(4), 25 (2007)
19. Fagin, R., Kolaitis, P.G., Miller, R.J., Popa, L.: Data Exchange: Semantics and Query Answering. Theoretical Comput. Sci. 336(1), 89–124 (2005)
20. Fagin, R., Kolaitis, P.G., Nash, A., Popa, L.: Towards a theory of schema-mapping optimization. In: Proceedings of the ACM Symposium on Principles of Database Systems (PODS), pp. 33–42 (2008)
21. Fagin, R., Kolaitis, P.G., Popa, L., Tan, W.: Composing schema mappings: Second-order dependencies to the rescue. ACM Transactions on Database Systems (TODS) 30(4), 994–1055 (2005)

22. Fagin, R., Kolaitis, P.G., Popa, L., Tan, W.-C.: Quasi-inverses of schema mappings. ACM Transactions on Database Systems (TODS) 33(2), 1–52 (2008)
23. Franklin, M.J., Halevy, A.Y., Maier, D.: From databases to dataspaces: a new abstraction for information management. SIGMOD Record 34(4), 27–33 (2005)
24. Fuxman, A., Hernández, M.A., Ho, H., Miller, R.J., Papotti, P., Popa, L.: Nested Mappings: Schema Mapping Reloaded. In: Proceedings of the International Conference on Very Large Data Bases (VLDB), pp. 67–78 (2006)
25. Fuxman, A., Kolaitis, P.G., Miller, R., Tan, W.-C.: Peer Data Exchange. ACM Transactions on Database Systems (TODS) 31(4), 1454–1498 (2006)
26. Haas, L.: Beauty and the beast: The theory and practice of information integration. In: Schwentick, T., Suciu, D. (eds.) ICDT 2007. LNCS, vol. 4353, pp. 28–43. Springer, Heidelberg (2006)
27. Haas, L.M., Hernández, M.A., Ho, H., Popa, L., Tork Roth, M.: Clio grows up: From research prototype to industrial tool. In: ACM SIGMOD Conference, pp. 805–810 (2005)
28. Halevy, A.Y., Ives, Z.G., Madhavan, J., Mork, P., Suciu, D., Tatarinov, I.: The piazza peer data management system. IEEE Transactions On Knowledge and Data Engineering 16(7), 787–798 (2004)
29. Hernández, M.A., Papotti, P., Tan, W.-C.: Data exchange with data-metadata translations. Proceedings of the VLDB Endowment 1(1), 260–273 (2008)
30. Hull, R., Yoshikawa, M.: ILOG: Declarative Creation and Manipulation of Object Identifiers. In: Proceedings of the International Conference on Very Large Data Bases (VLDB), pp. 455–468 (1990)
31. Jiang, H., Ho, H., Popa, L., Han, W.-S.: Mapping-driven XML transformation. In: Proceedings of the International WWW Conference, pp. 1063–1072 (2007)
32. Jiang, L., Borgida, A., Mylopoulos, J.: Towards a compositional semantic account of data quality attributes. In: Li, Q., Spaccapietra, S., Yu, E., Olivé, A. (eds.) ER 2008. LNCS, vol. 5231, pp. 55–68. Springer, Heidelberg (2008)
33. Lenzerini, M.: Data Integration: A Theoretical Perspective. In: Proceedings of the ACM Symposium on Principles of Database Systems (PODS), pp. 233–246 (2002)
34. Levy, A.Y., Rajaraman, A., Ordille, J.J.: Querying Heterogeneous Information Sources Using Source Descriptions. In: Proceedings of the International Conference on Very Large Data Bases (VLDB), pp. 251–262 (1996)
35. Madhavan, J., Halevy, A.Y.: Composing Mappings Among Data Sources. In: Proceedings of the International Conference on Very Large Data Bases (VLDB), pp. 572–583 (2003)
36. Maier, D., Mendelzon, A.O., Sagiv, Y.: Testing Implications of Data Dependencies. ACM Transactions on Database Systems (TODS) 4(4), 455–469 (1979)
37. Melnik, S., Bernstein, P.A., Halevy, A., Rahm, E.: Applying model management to executable mappings. In: ACM SIGMOD Conference, pp. 167–178 (2005)
38. Miller, R.J., Haas, L.M., Hernández, M.: Schema Mapping as Query Discovery. In: Proceedings of the International Conference on Very Large Data Bases (VLDB), pp. 77–88 (2000)
39. Milo, T., Zohar, S.: Using Schema Matching to Simplify Heterogeneous Data Translation. In: Proceedings of the International Conference on Very Large Data Bases (VLDB), pp. 122–133 (1998)
40. Nash, A., Bernstein, P.A., Melnik, S.: Composition of mappings given by embedded dependencies. In: Proceedings of the ACM Symposium on Principles of Database Systems (PODS), pp. 172–183 (2005)

41. Papakonstantinou, Y., Abiteboul, S., Garcia-Molina, H.: Object fusion in mediator systems. In: Proceedings of the International Conference on Very Large Data Bases (VLDB), pp. 413–424 (1996)
42. Popa, L., Tannen, V.: An Equational Chase for Path-Conjunctive Queries, Constraints, and Views. In: Beeri, C., Bruneman, P. (eds.) ICDT 1999. LNCS, vol. 1540, pp. 39–57. Springer, Heidelberg (1998)
43. Popa, L., Velegrakis, Y., Miller, R.J., Hernández, M.A., Fagin, R.: Translating Web Data. In: Proceedings of the International Conference on Very Large Data Bases (VLDB), pp. 598–609 (2002)
44. Raffio, A., Braga, D., Ceri, S., Papotti, P., Hernández, M.A.: Clip: a Visual Language for Explicit Schema Mappings. In: Proceedings of International Conference on Data Engineering (ICDE), pp. 30–39 (2008)
45. Salles, M.A.V., Dittrich, J.-P., Karakashian, S.K., Girard, O.R., Blunschi, L.: iTrails: Pay-as-you-go information integration in dataspaces. In: Proceedings of the International Conference on Very Large Data Bases (VLDB), pp. 663–674 (2007)
46. Shu, N.C., Housel, B.C., Lum, V.Y.: Convert: A high level translation definition language for data conversion. Commun. ACM 18(10), 557–567 (1975)
47. Shu, N.C., Housel, B.C., Taylor, R.W., Ghosh, S.P., Lum, V.Y.: EXPRESS: A Data EXtraction, Processing and REstructuring System. ACM Transactions on Database Systems (TODS) 2(2), 134–174 (1977)
48. Velegrakis, Y.: Managing Schema Mappings in Highly Heterogeneous Environments. PhD thesis, Department of Computer Science, University of Toronto (2004)
49. Velegrakis, Y., Miller, R.J., Popa, L.: On Preserving Mapping Consistency under Schema Changes. International Journal on Very Large Data Bases 13(3), 274–293 (2004)
50. Wyss, C.M., Robertson, E.L.: Relational languages for metadata integration. ACM Transactions on Database Systems (TODS) 30(2), 624–660 (2005)
51. Yan, L.-L., Miller, R.J., Haas, L., Fagin, R.: Data-Driven Understanding and Refinement of Schema Mappings. ACM SIGMOD Conference 30(2), 485–496 (2001)
52. Yu, C., Popa, L.: Constraint-Based XML Query Rewriting For Data Integration. ACM SIGMOD Conference 33(2), 371–382 (2004)

Heterogeneity in Model Management: A Meta Modeling Approach

Matthias Jarke[1,2], Manfred A. Jeusfeld[4], Hans W. Nissen[2,3], and Christoph Quix[1]

[1] RWTH Aachen University, Informatik 5, Ahornstr. 55, 52074 Aachen, Germany
[2] Fraunhofer FIT, Schloss Birlinghoven, 53754 Sankt Augustin, Germany
[3] Cologne University of Applied Sciences, Betzdorfer Str.2, 50679 Köln, Germany
[4] Tilburg University, The Netherlands
Jarke@cs.rwth-aachen.de

Abstract. As models are always abstractions of reality, we often need multiple modeling perspectives for analysis. The interplay of such modeling perspectives can take many forms and plays a role both at the design level, and during the operation of information systems. Examples include viewpoint resolution in requirements management, mapping between conceptual and implementation design in databases, and the integration or interoperation of multiple data and media sources. Starting from early experiences with our now 20-year old ConceptBase implementation of the Telos language, we describe a logic-based conceptual modeling and model management approach to these issues, focusing on recent work which employs a generic meta model to facilitate mappings and transformations between heterogeneous model representations both at the schema and the data level.

1 Introduction

Only a few years after the introduction of conceptual modeling via early representatives such as the Entity-Relationship Model or Structured Analysis and Design, John Mylopoulos began a research agenda of investigating conceptual models not just as pretty pictures for communication, but as formal objects with a logic foundation. Exploiting the parallel advent of logic programming and logic databases, models thus became accessible to, and useful for the fields of (a) Artificial Intelligence (e.g. reasoning about model consistency), (b) Databases (e.g. storing, retrieving, and integrating models; using models as organizing principles for large data sets), and (c) Software Engineering (e.g. model-driven code generation, lifecycle modeling and traceability, version and configuration management) [7].

Well-known early instances of this research agenda were the database design language Taxis [34] and the requirements modeling language RML [16]. However, the host of object-oriented modeling formalisms in the mid-1980s and early precursors of model-driven architectures such as DAIDA [6, 24] demonstrated that it was not enough to formalize individual modeling notations. With equal importance, the relationships between models had to be managed. Examples include model transformations and code generation, but also mappings among existing models, for schema integration or traceability of design artifacts [43].

To avoid pairwise mappings between all the available modeling notations, the idea of *meta modeling* was pursued by a number of researchers and standardization bodies. As data can be seen as instances of the corresponding schemas (in short: models), these models themselves can be seen as instances of one or more meta models. A meta model can reflect generic knowledge about the language notation or the domain ontology in which the model is expressed. The issue of relationships arises again between meta models, such that in principle an infinite regress of meta meta models (M2 models), meta meta meta models (M3 models), etc. emerges.

The different approaches to meta model management have addressed this issue in different ways. Pioneered by the Information Resource Dictionary Standard [IRDS90], most approaches stop at the M^2 level, arguing that a simple generic meta meta model is sufficient to express the notations and relationships. Typically, this M^2 model reflects an elementary graph construct (e.g. in MetaEdit+ [28]), or the basic class constructs such as generalization and aggregation (e.g. in the UML meta model and the OMG Meta Object Facility [39]). In some cases, a richer generic meta model, usually a domain-specific one, is employed to facilitate a higher quality of consistency checking, or a higher degree of automated code generation.

The four-level meta model architecture comprises the data layer, the schema or model layer, the meta model layer, and the M2 layer. For example, `John` can be considered an instance of the model class `Person` which is an instance of the meta model object `Entity` which is an instance of the M2 model object `Class`.

A related issue is the question how these levels are related to each other, i.e. the formal semantics of this architecture. For example, one might want to express the constraint that an instance of `Person` must be linked to at most one `Social Security Number`, or that an instance of `Entity` may define mandatory `required` relationships for all its instances, or even that any design object, irrespective of its notation, should be traceable to some instance of an instance of meta model object `Requirement`.

In IRDS, whose formalization was based on the SQL relational database standard, the four levels are organized in three interlocking level pairs, which represent application databases, data dictionaries or repositories, and method engineering environments [22]. In each pair, the schema will express (SQL) structures, queries, and integrity constraints about the level below. Analogously, in UML, an Object Constraint Language OCL was added in which constraints about a model can be expressed in meta models. Unfortunately, none of these approaches permits the specification of constraints across multiple levels of abstraction, such as e.g. the above traceability constraint. We call such multi-level constraints *meta formulas*.

The *Telos* language developed by an international network of researchers around John Mylopoulos in several iterations during the second half of the 1980's was not only one of the first, but also perhaps the most radical attempt at meta modeling. *Telos* does not only allow the specification of an arbitrary number of meta levels, but also the specification of meta formulas. Since this demands, in principle, second-order logic to be dealt with, the specification of *Telos* syntax and semantics had to be restricted with great care, in order to maintain computational tractability. A first paper-based formalization of *Telos* was completed in 1988 by Manolis Koubarakis, adding an important component to the fundamental *Telos* paper written during the sabbatical of Matthias Jarke in Toronto in late 1988 [25]. Shortly afterwards, based on results from the European Computational Logic initiative Compulog, Manfred Jeusfeld succeeded in defining a simpler semantics on the basis of Datalog with stratified negation with only very minor restrictions of expressiveness [26]. This formalization allowed the power of logic programming technologies such as partial evaluation to be applied to the *Telos* implementation and provided, for the first time, query and constraint optimization of meta formulae based on deductive database technologies [JeJa90]. Our ConceptBase deductive object base, whose first version was completed already in 1988 [25], was extended by these mechanisms [20] and has since been further developed for many model management tasks.

2 Meta Formulae in ConceptBase

Perhaps the most distinguishing property of *Telos* is its ability to represent concepts at any abstraction level (data, classes, meta classes, M^2 classes) uniformly with a single quadruple data structure called proposition: `P(o,x,n,y)`. Comparable to RDF, this data structure reifies all non-derived attributes, instantiations, and subclass relations.

Correct attribution, classification and specialization is expressed in logical axioms. ConceptBase realizes them via a mapping to Datalog with stratified negation. They form the pre-defined set of deductive rules and integrity constraints. As *Telos* allows the definition of new (generic) constructs at the meta and metameta class level; the meaning of the new constructs can be constrained by Datalog rules as well.

Traditionally, Datalog rules are only expressed for level pairs, i.e. for two consecutive abstractions levels of the Meta Object Facility MOF. Fig. 1 shows an example incarnation of the four MOF abstraction levels.

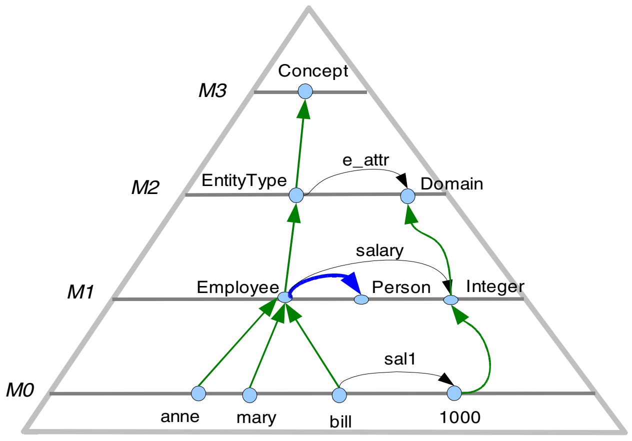


Fig. 1. Telos example in the MOF hierarchy

An example rule at level M1 could be that any employee has at least one salary:

$$\forall\ e\ In(e,Employee) \Rightarrow \exists\ s,n\ In(s,Integer) \wedge A(e,salary,n,s)$$

The predicate In(x,c) denotes that x is an instance of c. The predicate A(x,m,n,y) denotes that x has an attribute with label n and attribute category m and value y. The integrity constraint is displayed in first order logic but can be transformed to Datalog.

If some other attribute needs to be regarded as required, a similar constraint would have to be defined. To enable reuse of such formula patterns, we define the notion of a meta formulas, for example:

$$\forall c,d,m\ E(c,d,m) \Rightarrow (\forall x\ In(x,c) \Rightarrow \exists y,n\ In(y,d) \wedge A(x,m,n,y))$$
$$\forall a,c,m,d\ In(a,required) \wedge P(a,c,m,d) \Rightarrow E(c,d,m)$$

The first formula has a driver predicate E(c,d,m), which lists the so-called meta variables in the remainder of the formula. A meta variable is a variable c that occurs at the class position in In(x,c), or a variable m that occurs at the category position in A(x,m,n,y). A meta formula is then a formula with at least one meta variable. Using special predicates (x [in] c) and (x [m] y), the ConceptBase user can ask directly for instances of instances x of a class c, respectively for the values y of all instances of attribute metaclass m of a given object x.

It can be shown that any range-restricted formulas can be translated to one of the following two normal forms:

$$\forall\ \underline{c}\ E(\underline{c}) \Rightarrow \mathit{er}(\underline{c})$$
$$\exists\ \underline{c}\ E(\underline{c}) \wedge \mathit{er}(\underline{c})$$

This allows ConceptBase to apply a partial evaluation technique to manage meta formulas such as the one for the 'required' construct. ConceptBase maintains the extension of the driver predicate E and replaces E(c) in the meta formula by its extension. This generates a simplified version of the original meta formula where all meta variables are eliminated. If the meta formula was universally quantified, the simplified version decomposes into a conjunction of Horn clauses, ready to be interpreted by a Datalog engine.

Meta formulae can be equally well applied to formalize *generic constructs* such as transitivity, symmetry etc. These generic constructs can then be used to define the meaning of sub-classing, part-of relations, but also of traceability. A library of predefined meta formulae simplifies the definition of new constructs at the M2 level.

In the remainder of this chapter, we demonstrate the versatility of this approach by three examples. First, we present a rather direct application where the meta model hierarchy and a goal-oriented usage of meta formulae are employed to identify and manage inconsistencies in requirements specifications in a productive manner. Second, we discuss model-based data integration in the context of data quality management in data warehouses; here, the specification of inter-model relationships also needs to be propagated to the data level. Finally, we discuss how some of the key ideas of generic, logic-based meta models can be extended to allow the management of multiple pre-existing, complex information models and peer data networks expressed in heterogeneous formalisms.

3 Perspective Resolution in Requirements Engineering

Systems analysis and design methodologies for information systems employ multiple partial models or perspectives to structure the set of requirements specifications and designs. Since the early 1990's, authors from the software engineering community noticed that multiple stakeholders pursue conflicting opinions, have contradicting requirements and alternative perspectives [3, 13, 11].

In consulting practice, such approaches have been pursued since the late 1980s, albeit without much computer support. Prominent examples include IBM's JAD (Joint Application Design) [2], SSM (Soft Systems Methodology) [8], and PFR (Analysis of Presence and Future Requirements) [1]. Such goal-oriented teamwork approaches specifically follow the objective to capture requirements from all available sources and to make arising conflicts productive for deeper understanding and requirements negotiation. Their main characteristics are: informal and teamwork-oriented information acquisition, no fixed set of notations, goal-oriented perspective analysis, and conflict tolerance.

Our approach for a metamodel-based perspective resolution in requirements engineering is presented in Fig. 2 [36]. In the following paragraphs we explain the five features indicated by the numbers in the figure in more detail.

Separation of Multiple Perspectives (1). The conceptual models represent individual perspectives of stakeholders. The figure shows three perspectives (`A1,A2,A3`) expressed in `Notation A` and two perspectives (`B1,B2`) expressed in `Notation B`. *Telos* modules offer independent modeling contexts and enables

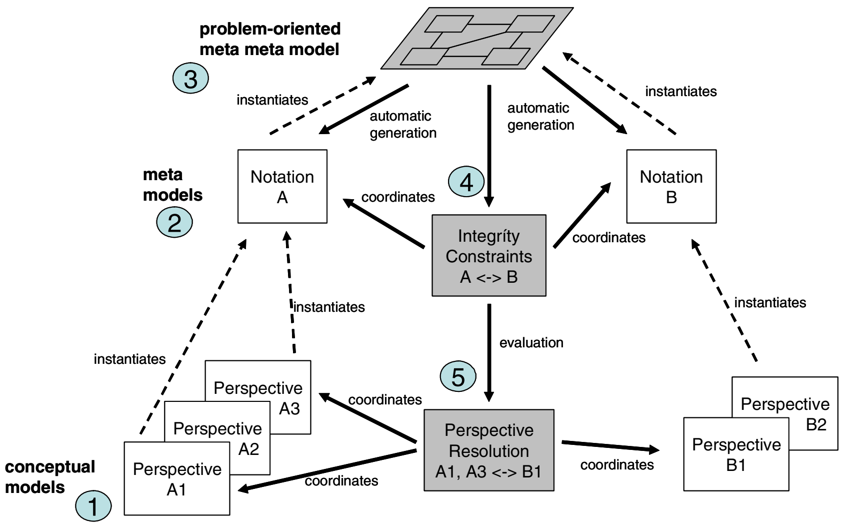


Fig. 2. Meta model driven perspective resolution

the representation of inconsistent conceptual models l; in ConceptBase, modules also contain an authorization function for security. The intended application scenario of modules in concurrent conceptual modeling processes induces the need for communication between modules [38]. Often, one modeling task depends on another one and reuses a part of its results. To support this situation, two modules can communicate by setting up an import-relationship. The importing module obtains access to the contents of the imported module. To protect a specific part of the module contents, the concept allows the division of the accessible contents of a module into a private and a public part. We had to add only seven new rules and six new constraints to the *Telos* definition to introduce modules. In addition a number of pre-defined objects were defined in order to introduce the built-in module `System` and its relationships to other pre-defined objects. The full set of axioms and pre-defined objects is given in [36]

Customizable Notations (2). We enable customizable notations by employing the extensible meta modeling capability of *Telos*. The language allows to define *and* modify meta models of the desired notations. The notations used to express the partial models are specified on the second level, the meta level. The example comprises two notations, `Notation A` and `Notation B`.

Adaptable Specification of Analysis Goals (3). A shared M2 model inter-relates the modeling notations. It specifies the domain structure as well as specific analysis goals. This model is created in teamwork at the beginning of the analysis project and documents an agreed shared domain ontology of all participating stakeholders. An analysis goal is actually an integrity constraint which defines a cross-perspective check performed on the bottom level between concrete conceptual models.

Goal-Oriented Perspective Analysis (4). The analysis goals specified in the problem-oriented meta meta model are stated independently from any specific notation. They are formulated exclusively on the domain ontology. In order to be able to analyze the partial models on the bottom layer, we transform the analysis goals into integrity constraints on the notation meta models. If an analysis goal covers two or more notations, the integrity constraint will be placed in a special module, the resolution module. In such a module, all integrity constraints which specify the relationships between partial models of the connected notations are collected.

Tolerance of Conflicts (5). To avoid interrupting the creative modeling activity in the presence of inter-perspective inconsistencies, the cross-perspective analysis takes place in separate resolution modules. The figure shows such a resolution module to check the perspectives `A1,A3` and `B1`. If the analysis results in a conflict between perspectives, the conflict is continuously documented but it is not required that the conflict is resolved immediately. Since the documentation is visible only within the resolution module, the designers of the involved models are not affected.

The tolerance of conflicts within a resolution module is possible due to a relaxed consistency enforcement approach. We allow insertions and deletions with a resolution module even if they cause an inconsistency. But a totally anarchic state of the object base is not desired. The inconsistent objects are managed in the sense that the object base knows about the inconsistent objects it contains. We have developed specialized inconsistency detection and management extensions to tolerate conflicts but keep the formal semantics of the object base.

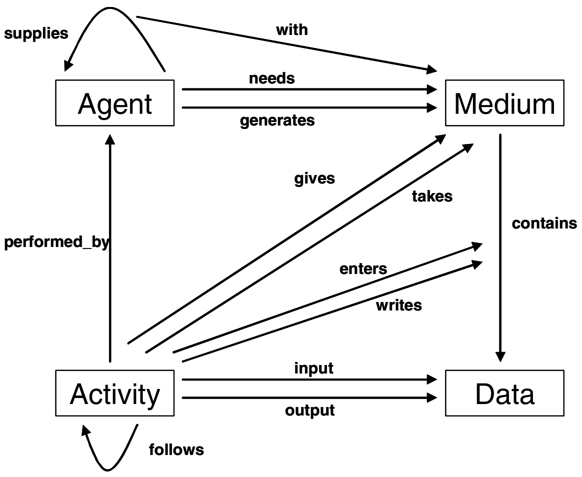


Fig. 3. The PFR meta meta model

Fig. 3 shows a *Telos* M2 model that has been used in several commercial analysis projects using the PFR method [37]. The conceptual models are analyzed from three perspectives: *information-exchange, activity-sequence*, and *document-structure*. The information-exchange perspective is represented by an `Agent` who `supplies` other agents `with a Medium`, the activity-sequence by the `Activity` that is `performed_by` an `Agent` and produces `Data` as `input` or `output`, and the document-structure by a `Medium` that `contains Data`. The M2 model contains a precise description of the terms employed during a PFR analysis. Its structure focuses on the expected problems in the domain. The distinction between `Medium` and `Data`, for example, is essential to talk about the unnecessary exchange of documents, i.e. documents which contain data that is not used by any activity.

Further experiences with this approach were gained in subsequent work in the international Tropos project where we analyze the design and operation of organization networks, by linking meta models such as i* strategic goals and dependences with workflow models e.g. based on speech act theory [15].

Our meta model-centered approach can be contrasted with more direct implementations of the ViewPoint approach [38]. ViewPoints aim at a completely distributed architecture without any central control unit or method engineering environment. Relationships between ViewPoints are defined by direct inter-ViewPoint rules without overarching analysis goals. In addition, the *Telos* approach allows the incorporation of customized notations, even at analysis time.

4 Model-Based Information Integration in Data Warehouses

While perspective resolution in requirements engineering typically operates at the modeling level only, the design and operation of data warehouses has to consider the integration and presentation of existing data, in addition to the schemas. We have studied this problem in the European DWQ project [23].

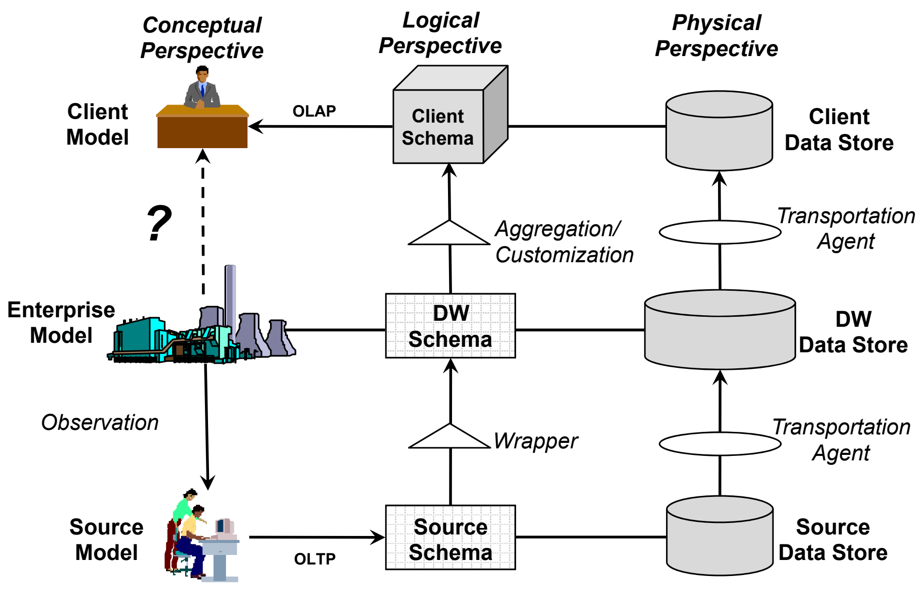


Fig. 4. Quality-Centered Data Warehouse Design in the DWQ project [JJQV99]

Traditionally, a data warehouse has been seen as a collection of views on various operational data sources, typically collected over an extended period of time and with a higher level of aggregation than the original data. This should enable online analytic processing (OLAP) by the client in some kind of multi-dimensional data model. In other words, the “global” data warehouse is seen as a view on the data sources which again provides specialized views (data marts) for various analysis purposes.

However, this Global-as-View (GAV) approach may not be appropriate, if multiple perspectives on the same reality and data quality need to be taken into account. As Fig. 4 illustrates, the analyst wants to study a reality (say, an enterprise), which he cannot observe directly. Different observations (documented transactions) have been collected in data sources about this reality, such that these data sources must in principle be seen as local views on the reality which should be represented by the data warehouse. This Local-as-View (LAV) approach allows the data warehouse designer to specify quality issues such as completeness, actuality, and precision of the data sources with respect to an “ideal” enterprise information system, thus warning the analyst of gaps where perhaps no data or only outdated or imprecise data exist.

The LAV approach requires, similar to the previous section, that the data warehouse conceptual model and schema is initially specified independently of those of the available data sources. Incidentally, this goes along well with current conceptual modeling practice in industry which is often organized around a business process model independent of available information systems. Inter-model constraints between the data warehouse model and the models of the data sources are then specified separately, in a manner similar to the one described in the previous section. Within a given modeling formalism, powerful description logic formulae can be used for the specification of these relationships [32] but for a setting of heterogeneous formalisms a homogenization using a language such as Telos is first needed.

While there are clear advantages to the LAV approach in terms of data quality management and understandability for the analyst, the mapping from the schema level to the data level is significantly more complicated than in the GAV approach which just involves regular simple querying of the sources using traditional view unfolding. For LAV, an approach to question-answering using a pre-defined set of views (the data sources) is required; a number of such approaches have appeared in the last decade [17]. In our DWQ implementation, the inter-schema constraints have been mapped down to a variant of the MiniCon algorithm [40, 41].

5 Dealing with Model Complexity: Model Management

5.1 The Model Management Movement

Referring to early experiences as described above, Bernstein et al. [4] brought up an additional problem : the management of very large-scale legacy schemata and models which can no longer be handled manually, yet need to be manipulated easily even by relatively inexperienced users. This new approach was called “model management” (not to be confused with mathematical model management discussed since the mid-1980s in Operations Research and Computational Engineering Sciences). The main idea is that, in a model management system, models and mappings should be considered as first class objects and that operations should address models as a whole and not only one model element at a time.

Model management aims at developing technologies and mechanisms to support the integration, merging, evolution, and matching of data models at the conceptual and logical design level. Bernstein et al. [4] proposed an algebraic approach with a few well-defined operators: `Match` for identifying relationships and the generation of a mapping between two schemas; `Merge` for the integration of schemas; and `Compose` for the composition of mappings. Other operators such as `DeepCopy` and `Diff` were defined in order to support complex sequences of model management operations which could be covered completely by the algebra. While such operations had already been discussed in the literature before, the aim was to generalize and integrate the existing approaches in a unified, generic model management system.

Several prototypes based on this approach were built, but the initial goal of having a set of independent operations which could be combined in arbitrary ways turned out to be too challenging, largely because the mapping representation had been underestimated. Although there have been many proposals for automatic methods for schema matching [42, 44], the task of defining a complete and correct mapping between two models still requires a lot of human effort, as the result of a match operator can be only an approximation of the correct mapping. In addition, mappings come in different flavors. For matching and simple schema integration tasks, pair-wise correspondences of schema elements might be sufficient; however, more formal tasks such as data translation or the integration of complex schemas require more expressive mapping languages. Finally, the definition of a mapping might depend on the context where it should be used. *Extensional* mappings refer data instances and can be used for data translation from one data source to another one, *intensional* mappings refer to the relationships of the schemas at the schema level, i.e. the intended meaning of a schema element.

Because of the observation that the definition of mappings and their application is a key issue in model management, Bernstein and Melnik [5] revised the initial vision to put mappings into the focus. Similar to the discussion in the data warehouse setting, mappings need to be expressed as queries. However, mappings between heterogeneous formalisms again need the genericity of meta modeling. Therefore, we developed a generic meta model which allows a uniform representation of data models, regardless of the modeling language which was used originally to define the data model. For a number of reasons, we do not rely on the *Telos* formalism as such, but extract some of the most important meta formulas and embed them in a generic meta model which is represented graphically in a UML-like notation.

The generic meta model *GeRoMe* [29] employs a role-based modeling approach in which an object is regarded as playing roles in collaborations with other objects. This allows describing the properties of model elements as accurately as possible while using only metaclasses and roles from a relatively small set. Therefore, *GeRoMe* provides a generic, yet detailed representation of data models which supports also the definition of mappings using an expressive generic mapping language based on second-order tuple generating dependencies, supported by our model management toolkit *GeRoMeSuite* [30].

5.2 The Generic Role-Based Metamodel GeRoMe

Most existing systems on model and mapping management deals only with one or two modeling languages (e.g. Clio [18]) or uses a rather simple generic representation based on graphs (e.g. Rondo [33]), in which case the detailed semantics behind the models is lost. The goal of *GeRoMe* is a detailed and uniform representation of data models, enabling the integration, transformation, and mapping of models across multiple formalisms. The most important role classes of the *GeRoMe* meta model are shown in Fig. 5.

In *GeRoMe* each model element of a native model (e.g. an XML schema or a relational schema) is represented as an object that plays a set of roles which decorate it with features and act as interfaces to the model element. Fig. 7 shows an example of a *GeRoMe* model of the XML Schema shown in Fig. 6.

The gray boxes in Fig. 7 denote model elements, the attached white boxes represent the roles played by the model elements. XML Schema is in several aspects different from “traditional” modeling languages such as EER or the Relational Metamodel. The main concept of XML Schema “element” represents actually an association between the parent complex type and the nested type. This is true for all elements except those that are allowed as the root element of a document. In *GeRoMe*, the definition of a root element is an association between the schema node and the element's complex type, as there is no complex type in which the root element is nested. In the example, the element `University` is an association between the model element `Schema` and the complex type `UniType`. The fact that the `University` element is an association is described by the *Association* (As) role which connects the *ObjectSet* (OS) roles of `Schema` and `UniType` via two anonymous model elements playing a *CompositionEnd* (CE) and an *ObjectAssociationEnd* (OE) role, respectively.

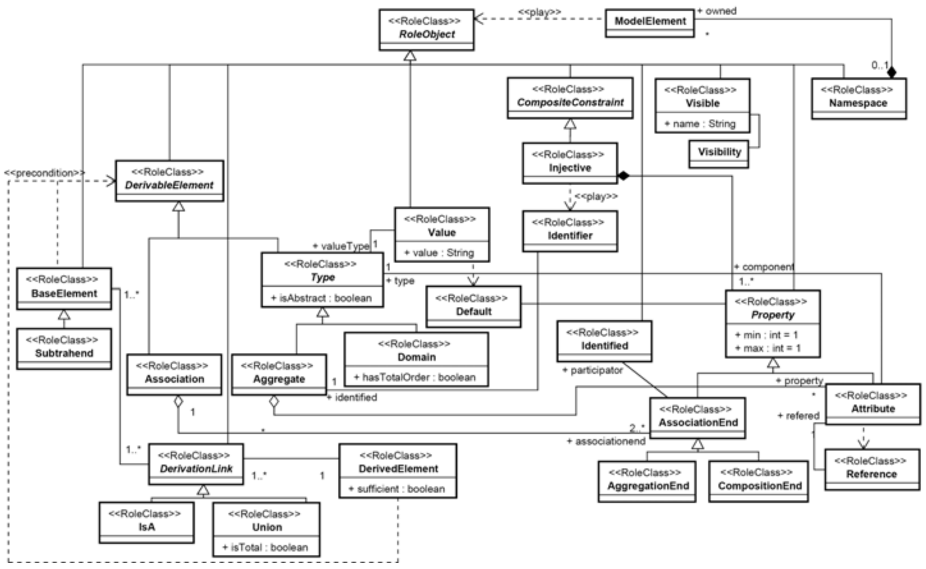


Fig. 5. Subset of the *GeRoMe* meta model

```
<xsd:schema>
  <xsd:element name="University" type="UniType"/>
  <xsd:complexType name="UniType">
    <xsd:sequence>
      <xsd:element name="Student" type="StudType"/>
    </xsd:sequence>
    <xsd:attribute name="uname" type="xsd:string"/>
  </xsd:complexType>
  <xsd:complexType name="StudType">
    <xsd:attribute name="sname" type="xsd:string"/>
    <xsd:attribute name="ID" type="xsd:integer"/>
  </xsd:complexType>
</xsd:schema>
```

Fig. 6. XML Schema for universities and students

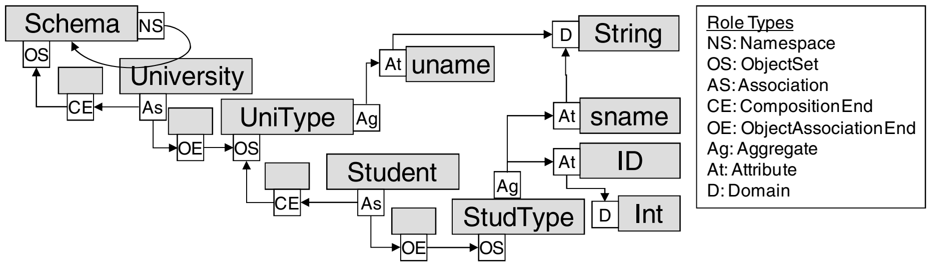


Fig. 7. *GeRoMe* representation of the XML schema in Fig. 6

Since *GeRoMe* is a *generic* meta model, we must take care that comparable modeling constructs from different modeling languages are represented uniformly. For instance, in UML, associations may be of degree higher than two, whereas in the ontology language OWL or in XML, instances of types (or classes) are only connected via binary associations. However, in order to allow uniform handling by model management operators, we need to represent all these types of associations in the same way, i.e. using associations and association end roles.

The same structure of association and association end roles is used for the element `Student` which is an association between the complex types `UniType` and `StudType`. The two complex types have attributes; therefore, they also play *Aggregate* (Ag) roles which link these model elements to their attributes. The model element representing attributes play *Attribute* (At) roles which refer also to the types of the attributes which are, in this example, simple domains denoted by the *Domain* (D) role.

Note that the representation of models in *GeRoMe* is not to be used by end users but internally by *GeRoMeSuite*, with the goal to provide more semantics to model management operators than a simple graph based model.

5.3 Generic Schema Mappings

The generic representation of models is the basis for the generic mapping language used in *GeRoMeSuite*. The mapping language is used to represent extensional relationships between models, i.e. between their instances. Therefore, we need to characterize instances of a *GeRoMe* model. Fig. 8 depicts an example instance for the *GeRoMe* model of the XML Schema in Figs. 6 and 7.

The first *inst*-predicate defines an instance of the schema element which represents the XML document itself. Two instances of the complex types and their attributes are defined using *av*-predicates, which state that an object has a certain *a*ttribute *v*alue for a specific attribute type. The last three lines define the associations and the relationships between the objects defined before. A predicate *part(as,ae,o)* denotes that the object *o* participates in the association *as* using the association end *ae*.

As the example shows, association end roles and attribute roles are not only able to define flat structures, but also nested structures, such as element hierarchies in XML schemas. This supports the representation of mappings between models with arbitrary structures as well as mappings which are able to restructure the data between the source and target model. We apply this basic idea to second order tuple generating dependencies (SO tgds) [12], which gives a rich expressive mapping language but still maintains features such as composability and executability.

```
<University uname="RWTH">
  <Student sname="John"
           ID="123"/>
</University>
```

$$
\begin{array}{l}
inst(\#0, \texttt{Schema}),\\
inst(\#1, \texttt{UniType}), av(\#1, \texttt{uname}, \texttt{'RWTH'}),\\
inst(\#2, \texttt{StudType}),\\
av(\#2, \texttt{sname}, \texttt{'John'}), av(\#2, \texttt{ID}, 123),\\
inst(\#3, \texttt{University}), inst(\#4, \texttt{Student}),\\
part(\#3, \texttt{parent}_U, \#0), part(\#3, \texttt{child}_U, \#1),\\
part(\#4, \texttt{parent}_S, \#1), part(\#4, \texttt{child}_S, \#2)
\end{array}
$$

Fig. 8. XML document and its representation as GeRoMe instance

$$\begin{aligned}
&\exists f, g \quad \forall o_0, o_1, o_2, o_3, o_4, u, s, i \quad inst(o_0, \texttt{Schema}) \land inst(o_2, \texttt{UniType}) \land \\
&inst(o_1, \texttt{University}) \land part(o_1, \texttt{parent}_U, o_0) \land part(o_1, \texttt{child}_U, o_2) \land \\
&inst(o_4, \texttt{StudType}) \land \\
&inst(o_3, \texttt{Student}) \land part(o_3, \texttt{parent}_S, o_2) \land part(o_3, \texttt{child}_S, o_4) \land \\
&av(o_2, \texttt{uname}, u) \land av(o_4, \texttt{sname}, s) \land av(o_4, \texttt{ID}, i) \rightarrow \\
&\quad inst(f(u), \texttt{University}) \land inst(g(i), \texttt{Student}), \\
&\quad av(f(u), \texttt{uname}, u) \land av(g(i), \texttt{sname}, s) \land av(g(i), \texttt{ID}, i) \land av(g(i), \texttt{uni}, u)
\end{aligned}$$

Fig. 9. Generic mapping from an XML schema to a relational schema

The main feature of our mapping language is that we use reification to describe data structures by introducing abstract individuals to denote the instances of model elements. In the example of Fig. 8, the abstract individuals were identified by #0, #1, #2, … Instead of describing these individuals by a single predicate (e.g. like a tuple in a relational database), we use a set of predicates to describe each feature of that object, similar to Telos. In combination with the ability to describe arbitrary structures, the use of reification makes the mapping language also suitable for semi structured data models such as XML.

A detailed presentation of the mapping language is beyond the scope of this paper, details can be found in [31]. Fig. 9 shows an example of a mapping between models originally represented in two different modeling languages, where the target model is a relational schema with two relations `University(uname)` and `Student(id,sname,uni)`.

The predicates in the conditional part of the rule correspond to the instance predicates shown in Fig. 8, now just with variables instead of constants. The variables o_0 to o_4 represent abstract identifiers, their function is to describe (implicitly) the structure of the source data that is queried for. In other approaches for mapping representation (e.g. [14]) such structures are represented by nesting different sub-expressions of a query. Although nested mappings are easier to read, they are strictly less expressive than SO tgds. In addition, several mapping tasks such as composition, inverting, optimization, and reasoning have to be reconsidered for nested mappings; for example, it is not clear how to compose nested mappings and whether the result composing two nested mappings can be expressed as a nested mapping.

Similarly to the abstract variables on the source side, the functions *f* and *g* represent abstract identifiers on the target side, and therefore describe the structure of the generated target data. Such abstract functions can be understood as Skolem functions which do not have an explicit semantics; they are interpreted by syntactical representation as terms. While the mapping is executed, these functions are used to create the appropriate nesting and grouping structures of the target data.

To describe the structure of the target data, it is important to know which values are used to identify an object. According to the definition of the relational schema, universities are identified by their name (*u*) and students by their ID (*i*); that is why we use *u* and *i* as arguments of the abstract functions *f* and *g*.

In addition to abstract functions, a mapping can also contain concrete ("normal") functions for value conversions or other types of data transformation. While executing a mapping, these functions must be actually evaluated to get the value which has to be inserted into the target.

The model management system *GeRoMeSuite* includes a mapping execution engine. The system is able to execute mappings between XML, relational and Java models. The generic query expression on the source side is translated into an executable query of the corresponding query language (e.g. XQuery or SQL), while on the target side, the resulting objects are created.

5.4 Application Example: Generic Peer Data Management

As an application of our generic model management approach, we consider peer data management systems (PDMS) in a heterogeneous setting.

In PDMS, each peer can act as a data source as well as a mediator in the network. Queries can be posed against any peer and are then rewritten to queries against relevant data sources. Applications of peer data management can be found in the areas of personal dataspaces and the semantic web. Both scenarios exhibit highly heterogeneous data sources which may not only differ in the schemas exposed, but also in the modeling languages used to describe the schemas. For instance, whereas some data sources may be relational, others can be based on XML Schema or ontologies. Such a PDMS requires a mapping language and query rewriting algorithms that can deal with different modeling languages. We applied our generic approach to query rewriting in PDMS. This includes rewriting of queries using global-as-view and local-as-view mappings.

Fig. 10 sketches an example of query processing in the PDMS which uses view unfolding and query folding for query rewriting. The user specifies the query using a query interface similar to SQL. The user query Q, in the figure posed over the schema S_2 of the peer P_2, is translated into a generic query expression using a syntax similar to the generic mapping language sketched above. The generic query is sent to the peer which checks whether the available mappings to other peers are relevant to the query. In the example sketched in Fig. 10, the query can be answered by peer P_1 and peer P_3. The schema S_1 of peer P_1 is expressed as GAV mapping, e.g. the schema elements of S_2 are expressed as a view on the elements of S1. Therefore, we can use view unfolding: the mapping M_{12} is composed with the query Q which results in the rewritten query Q_{S1}, i.e. the query Q in terms of the schema S_1.

On the other hand, the mapping between P2 and P3 is expressed as a LAV mapping from the viewpoint of peer P2. This means, that the elements of S3 are expressed as views on S2. In this case, we again need the MiniCon algorithm [40, 41] for query rewriting of LAV mappings.

Due to the reification in our mapping language, the number of predicates in a mapping (or query) is significantly higher than in usual relational mappings. As the complexity of the algorithms for query rewriting depends on the size of queries and mappings (i.e. the number of predicates), the reified style of our mappings increases the complexity. Therefore, it is important to keep the number of predicates as low as possible while the query rewriting algorithms are applied.

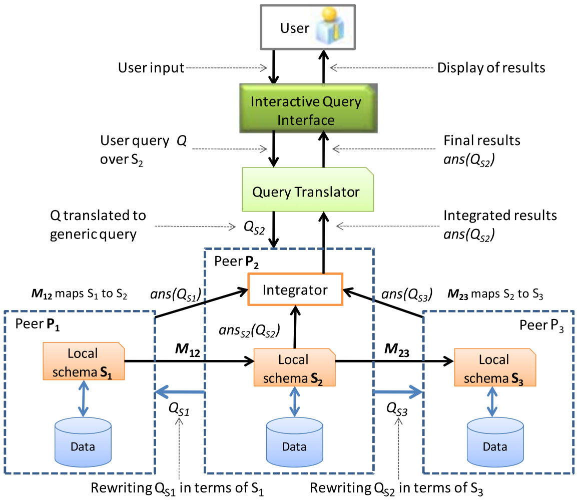

User
User input
Display of results
Interactive Query Interface
User query Q over S_2
Final results $ans(Q_{S2})$
Query Translator
Q translated to generic query
Q_{S2}
Integrated results $ans(Q_{S2})$
Peer P_2
Integrator
M_{12} maps S_1 to S_2
$ans(Q_{S1})$
$ans(Q_{S3})$
M_{23} maps S_2 to S_3
Peer P_1
Peer P_3
$ans_{S2}(Q_{S2})$
Local schema S_1
M_{12}
Local schema S_2
M_{23}
Local schema S_3
Q_{S1}
Q_{S3}
Data
Data
Data
Rewriting Q_{S1} in terms of S_1
Rewriting Q_{S2} in terms of S_3

Fig. 10. Query Processing in the Peer Data Management System

We have developed several optimization techniques inspired by the semantic optimization techniques applied in the ConceptBase system. The basic idea is to remove predicates which are already implied by other predicates [31].

6 Concluding Remarks

Heterogeneity of modeling notations, domain ontologies, and analysis goals is a fact of life in model management. In this chapter, we have demonstrated with case studies from requirements engineering, data warehouse management, and peer data management, how this heterogeneity can be exploited and managed through generic meta modeling technologies. The radical approach to meta modeling pioneered in the *Telos* language and its mapping to efficient Datalog code using a carefully designed, but extensible set of meta formulae turns out to be a key success factor here.

The fundamental quadruple structure underlying *Telos* which preceded, but also generalizes the triple structure underlying XML and, in particular, its metadata sublanguage RDF, is not only an important prerequisite for this solution but has also served as the basis for other important meta modeling standards beyond the examples given in this chapter. An important example is the CIDOC Conceptual Reference Model for model integration in digital libraries [10] which started from an implementation of a subset of *Telos* developed at the FORTH institute in Crete focusing just on its structural axioms [9]. Similar work linking conceptual modeling to the MPEG-7 and MPEG-21 multimedia standards in the context of Internet communities is currently pursued at RWTH Aachen University, extending the community analysis tools from [15] by media aspects.

References

1. Abel, P.: Description of the USU-PFR Analysis Method, Technical Report, USU GmbH, Möglingen, Germany (1995)
2. August, J.H.: Joint Application Design: The Group Session Approach to System Design. Yourdan Press, Englewood Cliffs (1991)
3. Balzer, R.: Tolerating Inconsistencies. In: 13th Intl. Conf. Software Engineering (ICSE-13), Austin, Texas, pp. 158–165 (1991)
4. Bernstein, P.A., Halevy, A.Y., Pottinger, R.A.: A vision for management of complex models. ACM SIGMOD Record 29(4), 55–63 (2000)
5. Bernstein, P.A., Melnik, S.: Model management 2.0: Manipulating richer mappings. In: ACM SIGMOD Intl. Conf. on Management of Data, Beijing, China, pp. 1–12 (2007)
6. Borgida, A., Jarke, M., Mylopoulos, J., Schmidt, J.W., Vassiliou, Y.: The software development environment as a knowledge base management system. In: Schmidt, J.W., Thanos, C. (eds.) Foundations of Knowledge Base Management (Xania Workshop), pp. 411–442. Springer Topics in Information Systems (1986)
7. Brodie, M.L., Mylopoulos, J., Schmidt, J.W.: On Conceptual Modelling – Perspectives from Artificial Intelligence, Databases, and Programming Languages. Springer, Heidelberg (1984)
8. Checkland, P.B.: Soft Systems Methodology. In: Rosenhead, J. (ed.) Rational Analysis for a Problematic World, pp. 71–100. John Wiley & Sons, Chichester (1989)
9. Constantopoulos, P., Jarke, M., Mylopoulos, J., Vassiliou, Y.: The software information base: a server for reuse. VLDB Journal 4(1), 1–43 (1995)
10. Doerr, M.: The CIDOC Conceptual Reference Model: an ontological approach to semantic interoperability of metadata. AI Magazine 24(3), 75–92 (2003)
11. Easterbrook, S.M.: Learning from inconsistencies, 8th Intl. Workshop on Software Specification and Design, Schloss Velen, Germany (1996)
12. Fagin, R., Kolaitis, P.G., Popa, L., Tan, W.C.: Composing schema mappings: Second order dependencies to the rescue. ACM Trans. Database Systems 30(4), 994–1055 (2005)
13. Feather, M.S., Fickas, S.: Coping with requirements freedoms. In: Intl. Workshop on the Development of Intelligent Information Systems, Niagara-on-the-Lake, Ontario, Canada, pp. 42–46 (1991)
14. Fuxman, A., Hernandez, M.A., Ho, C.T.H., Miller, R., Papotti, P., Popa, L.: Nested mappings: schema mapping reloaded. In: Proc. 32nd Intl. Conf. Very Large Data Bases (VLDB 2006), pp. 67–78. ACM Press, New York (2006)
15. Gans, G., Jarke, M., Kethers, S., Lakemeyer, G.: Continuous requirements engineering for organization networks: a (dis-)trust-based approach. Requirements Eng. J. 8(1), 4–22 (2003)
16. Greenspan, S., Borgida, A., Mylopoulos, J.: A requirements modelling language and its logic. Information Systems 11(1), 9–23 (1986)
17. Halevy, A.Y.: Answering queries using views: a survey. VLDB Journal 10(4), 270–294 (2001)
18. Hernandez, M.A., Miller, R.J., Haas, L.M.: Clio: A semi-automatic tool for schema mapping. In: ACM SIGMOD Conf., Santa Barbara, CA, p. 607 (2001)
19. ISO/IEC International Standard, Information Resource Dictionary System (IRDS) – Framework, ISO/IEC 10027 (1990)
20. Jarke, M., Eherer, S., Gallersdörfer, R., Jeusfeld, M.A., Staudt, M.: ConceptBase – a deductive object base for meta data management. J. Intelligent Information Systems 4(2), 167–192 (1995)
21. Jarke, M., Jeusfeld, M.A., Quix, C., Vassiliadis, P.: Architecture and quality in data warehouses: an extended repository approach. Information Systems 24(3), 229–253 (1999)
22. Jarke, M., Klamma, R., Lyytinen, K.: Meta modeling. In: Jeusfeld, M.A., Jarke, M., Mylopoulos, J. (eds.) Meta Modeling for Method Engineering, MIT Press, Cambridge (2009) (to appear)

23. Jarke, M., Lenzerini, M., Vassiliou, Y., Vassiliadis, P.: Fundamentals of Data Warehouses, 2nd edn. Springer, Heidelberg (2003)
24. Jarke, M., Mylopoulos, J., Schmidt, J.W., Vassiliou, Y.: DAIDA – an environment for evolving information systems. ACM Trans. Information Systems 10(1), 1–50 (1992)
25. Jarke, M., Rose, T.: Managing knowledge about information systems evolution. In: ACM SIGMOD Conf., Chicago, IL, pp. 303–311 (1988)
26. Jeusfeld, M.A.: Update Control in Deductive Object Bases, PhD Thesis, University of Passau (1992) (in German)
27. Jeusfeld, M.A., Jarke, M.: From relational to object-oriented integrity simplification. In: Delobel, C., Masunaga, Y., Kifer, M. (eds.) DOOD 1991. LNCS, vol. 566, pp. 460–477. Springer, Heidelberg (1991)
28. Kelly, S., Lyytinen, K., Rossi, M.: MetaEdit+ – a fully configurable multi-user and multi-tool CASE and CAME environment. In: Constantopoulos, P., Vassiliou, Y., Mylopoulos, J. (eds.) CAiSE 1996. LNCS, vol. 1080, pp. 1–21. Springer, Heidelberg (1996)
29. Kensche, D., Quix, C., Chatti, M.A., Jarke, M.: GeRoMe – a generic role based meta model for model management. In: Spaccapietra, S., Atzeni, P., Fages, F., Hacid, M.-S., Kifer, M., Mylopoulos, J., Pernici, B., Shvaiko, P., Trujillo, J., Zaihrayeu, I. (eds.) Journal on Data Semantics VIII. LNCS, vol. 4380, pp. 82–117. Springer, Heidelberg (2007)
30. Kensche, D., Quix, C., Li, X., Li, Y.: GeRoMeSuite: A system for holistic generic model management. In: 33rd Int. Conf. Very Large Data Bases (VLDB 2007), Vienna, Austria, pp. 1322–1325 (2007)
31. Kensche, D., Quix, C., Li, X., Li, Y., Jarke, M.: Generic schema mappings for composition and query answering. In: Data & Knowledge Engineering (2009) (to appear)
32. Lenzerini, M.: Data integration: a theoretical perspective. 21. In: ACM Symp. Principles of Database Systems (PODS), Madison, Wisconsin, pp. 233–246 (2002)
33. Melnik, S., Rahm, E., Bernstein, P.A.: Rondo: a programming platform for generic model management. In: ACM SIGMOD Intl. Conf. Management of Data, San Diego, CA, pp. 193–204 (2003)
34. Mylopoulos, J., Bernstein, P.A., Wong, H.K.T.: A language facility for designing interactive database-intensive applications. ACM Trans. Database Syst. 5(2), 185–207 (1980)
35. Mylopoulos, J., Borgida, A., Jarke, M., Koubarakis, M.: Telos – representing knowledge about information systems. ACM Transactions on Information Systems 8(4), 325–362 (1990)
36. Nissen, H.W., Jarke, M.: Repository support for multi-perspective requirements engineering. Information Systems 24(2), 131–158 (1999)
37. Nissen, H.W., Jeusfeld, M.A., Jarke, M., Zemanek, G.V., Huber, H.: Managing multiple requirements perspectives with metamodels. IEEE Software 13(2), 37–48 (1996)
38. Nuseibeh, B., Kramer, J., Finkelstein, A.: A framework for expressing the relationships between multiple views in requirements specifications. IEEE Trans. Software Eng. 20(10), 760–773 (1994)
39. Object Management Group: Meta Object Facility/MOF core specification version 2.0. OMG (2006)
40. Pottinger, R., Halevy, A.Y.: MiniCon: a scalable algorithm for answering queries using views. VLDB Journal 10(2-3), 182–198 (2001)
41. Quix, C.: Metadata Management for Quality-Oriented Information Logistics in Data Warehouse Systems (in German). Ph.D. Thesis, RWTH Aachen University, Germany (2003)
42. Rahm, E., Bernstein, P.A.: A survey of approaches to automatic schema matching. VLDB Journal 10(4), 334–350 (2001)
43. Ramesh, B., Jarke, M.: Reference models for requirements traceability. IEEE Trans. Software Eng. 27(1), 58–93 (2001)
44. Shvaiko, P., Euzenat, J.: A survey of schema-based matching approaches. In: Spaccapietra, S. (ed.) Journal on Data Semantics IV. LNCS, vol. 3730, pp. 146–171. Springer, Heidelberg (2005)

Associativity and Commutativity in Generic Merge

Rachel Pottinger[1] and Philip A. Bernstein[2]

[1] University of British Columbia
201-2366 Main Mall
Vancouver, BC Canada
rap@cs.ubc.ca
[2] Microsoft Research
One Microsoft Way
Redmond, WA, USA
philbe@microsoft.com

Abstract. A model is a formal description of a complex application artifact, such as a database schema, an application interface, a UML model, an ontology, or a message format. The problem of merging such models lies at the core of many meta data applications, such as view integration, mediated schema creation for data integration, and ontology merging. This paper examines the problem of merging two models given correspondences between them. In particular it concentrates on the associativity and commutativity of $Merge$, which are crucial properties if $Merge$ is to be composed with other operators.

1 Introduction

A model is a formal description of a complex application artifact, such as a database schema, an application interface, a UML model, an ontology, or a message format. One of John Mylopoulos's major themes has been the development and use of such models. To apply such models to design-time and run-time scenarios requires that model transformations. These transformations are performed by operators on models and on mappings between models, which is a major subject of our own research. This paper focuses on one such operator, $Merge$.

The problem of merging models lies at the core of many meta data applications, such as view integration, mediated schema creation for data integration, and ontology merging. In each case, two given models need to be combined into one. Because there are many different types of models and applications, this problem has been tackled independently in specific domains many times. In [9] we generalized that work by presenting a generic operator, $Merge$, that can be applied in all these contexts. In this paper we extend [9] by showing that generic $Merge$ is commutative and associative.

Combining two models requires first determining correspondences between the two models and then merging the models based on those correspondences.

Finding correspondences is called schema matching; it is a major topic of ongoing research and is not covered here; see [6,10] for recent summaries. Rather, we focus on combining the models after correspondences are established. We encapsulate the problem in an operator, *Merge*, which takes as input two models, A and B, and a mapping Map_{A_B} between them that embodies the given correspondences, and returns a third model that is the "duplicate-free union" of A and B with respect to Map_{A_B}. This is not as simple as set union because the models have structure, so the semantics of "duplicates" and duplicate removal may be complex. In addition, the result of the union can manifest constraint violations, called conflicts, that *Merge* must repair.

An example of the problems addressed by *Merge* can be seen in Figure 1. It shows two representations of *Actor*, each of which could be a class, concept, table, etc. The solid diamond-headed arrows represent containment. A mapping between A and B is shown by the double dashed lines (=). In this case, it seems clear that *Merge* is meant to collapse *A.Actor* and *B.Actor* into a single element, and similarly for *Bio*. Clearly, *A.ActID* should be merged with *B.ActorID*, but what should the resulting element be called? Should the merged model represent the actor's name as one element (*ActorName*), two elements (*FirstName* and *LastName*), three elements (*ActorName* with *FirstName* and *LastName* as children), or in some other way? These cases of differing representations between input models are called conflicts. For the most part, conflict resolution is independent of how A and B are represented. Yet most work on merging schemas concentrates on doing it in a data-model-specific way, revisiting the same problems for ER variations [11], XML [2], data warehouses [5], or semi-structured data [3]. Note that these works, like ours, consider merging only the models, not the instances of the models.

The similarities among these solutions offer an opportunity for abstraction. One important step in this direction was a schema merging and conflict resolution algorithm by Buneman, Davidson, and Kosky (hereafter BDK) [4]. Given a set of pair-wise correspondences between two models that have specialization and containment relationships, BDK give a formal definition of merge and show how to resolve a certain kind of conflict to produce a unique result.

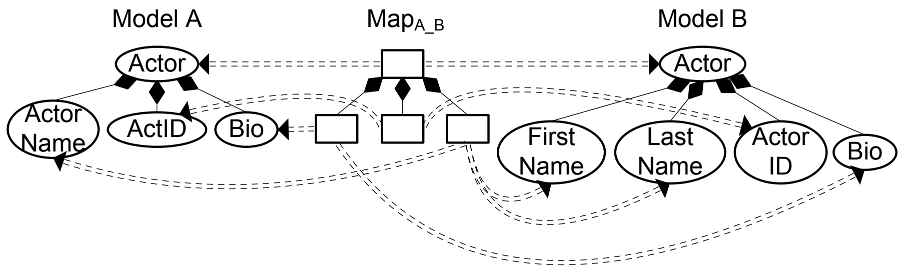


Fig. 1. Examples of models to be merged. The solid diamond-headed arrows represent containment. The double dashed lines (=) represent a mapping between A and B.

In [9] we presented a robust *Merge* algorithm based on BDK's algorithm by expanding the range of correspondences, model representations, conflict types, and applications. Our previous work did not consider the assocativity and commutativity of *Merge*. Understanding the associativity and commutivity of *Merge* is crucial to being able to compose *Merge* with other operators; therefore, this paper builds on [9] by analyzing the associativity and commutativity of our merge operator. Since this analysis refers heavily to the semantics and implementation of the operator, we review the main aspects of the approach in Sections 2 to 5 of this paper. Section 2 gives a precise definition of *Merge*. Section 3 describes our categorization of conflicts that arise from combining two models. Section 4 describes how to resolve conflicts in *Merge*, often automatically. Section 5 defines our merge algorithm. Section 6 presents the new material about the associativity and commutativity of *Merge*. Section 7 concludes.

2 Problem Definition

2.1 Representation of Models

Defining a representation for models requires (at least) three meta-levels. Using conventional meta-data terminology, we can have: a model, such as the database schema for a billing application; a meta-model, which consists of the type definitions for the objects of models, such as a meta-model that says a relational database schema consists of table definitions, column definitions, etc.; and a meta-meta-model, which is the representation language in which models and meta-models are expressed. For example, a generic meta-meta-model may say that a schema consists of objects, where an object could be a table, XML element, or a class definition.

The goal of our merge operator, *Merge*, is to merge two models based on a mapping between them. We discuss *Merge* using a small meta-meta-model consisting of:

1. Elements with semi-structured properties (i.e., for an element x, there may exist 0, 1, or many properties). Elements are the first class objects in a model. Three properties are required: *Name*, *ID*, and *History*. *Name* is self-explanatory. *ID* is the element's unique identifier. *History* describes the last operator that acted on the element. While all the elements in Figure 1 have *History* and *ID* properties, they are not shown for clarity.
2. Binary, directed, kinded relationships with cardinality constraints. A relationship is a connection between two elements. Relationships can be either explicitly present in the model or implied according to the meta-meta-model's rules. Such a rule might say that "a is a b" and "b is a c" implies that "a is a c." Relationship cardinalities are omitted from the figures for ease of exposition.

For concreteness, we will use our meta-meta-model Vanilla [9]. It contains 5 different relationship types:

– Associates - A(x, y) means x is associated with y. This is the weakest relationship that can be expressed. It has no constraints or special semantics.
– Contains - C(x, y) means container x contains containee y.
– Has-a - H(x, y) means x has a sub-component y (sometimes called "weak aggregation"). Has-a is weaker than Contains in that it does not propagate delete and can be cyclic.
– Is-a - I(x, y) means x is a specialization of y.
– Type-of - T(x, y) means x is of type y.

Figure 1 shows an example model in this small meta-meta-model; elements are shown as nodes, the value of the $Name$ property is the node's label, mapping relationships are edges with arrowheads, and containment relationships are diamond-headed edges.

2.2 *Merge* Inputs

The inputs to $Merge$ are the following:

1. Two models: A and B.
2. A mapping, Map_{A_B}, which is a model that defines how A and B are related.
3. An optional designation that one of A or B is the preferred model. When $Merge$ faces a choice that is not specified in the mapping, it chooses the option from the preferred model, if there is one.
4. Optional overrides for default $Merge$ behavior (explained further below).

The input mapping is more expressive than simple equalities; it is a first-class model consisting of elements and relationships. Some of its elements are mapping elements. A mapping element, x, is like any other element except it also is the origin of a mapping relationship, $M(x, y)$, which specifies that the origin element, x, represents the destination element, y. So a given mapping element, x, represents all elements y such that $M(x, y)$. All elements of Map_{A_B} in Figure 1 are mapping elements. In Map_{A_B} in Figure 3 $AllBios$ is not a mapping element.

There are two kinds of mapping relationships: equality and similarity. An equality mapping relationship Me asserts that for some set of models, collectively known as Y, for all elements $y1, y2 \in Y$ such that $Me(x, y1)$ and $Me(x, y2)$, $y1 = y2$. All elements represented by the same equality mapping relationship are said to correspond to one another. A similarity mapping relationship Ms asserts that the set of all $y \in Y$ such that $Ms(x, y)$ are related through the value of x's $Expression$ property, which is a property of all mapping elements that are the origin of mapping similarity relationships. For example, if $A.Bio$ is a French translation of $B.Bio$ and this needs to be reflected explicitly in the merged model, they could be connected by similarity mapping relationships to a mapping element (as shown in Figure 2) with an $Expression$ property "$A.Bio = English2French(B.Bio)$". The semantics of the $Expression$ property are not interpreted by $Merge$. Equality mapping relationships are represented by double-dashed-lines (=); similarity mapping relationships are represented by double-wavy-lines (≈).

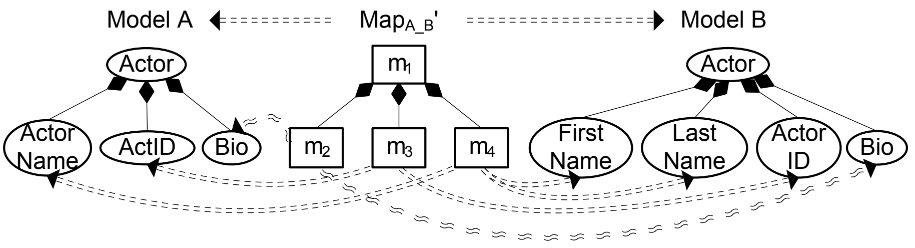


Fig. 2. A mapping using both equality mapping relationships (the double-dashed-lines) and similarity mapping relationships (the double-wavy lines)

Given this rich mapping structure, complex relationships can be defined between elements in A and B, not just simple correspondences. For example, the mapping in Figure 3 (which is between the same models in Figure 1) shows that the $FirstName$ and $LastName$ of model B should be child elements of the $ActorName$ element of model A; this is expressed by element $m4$, which represents $ActorName$ in A and contains elements $m5$ and $m6$ which represent $FirstName$ and $LastName$ respectively in B.

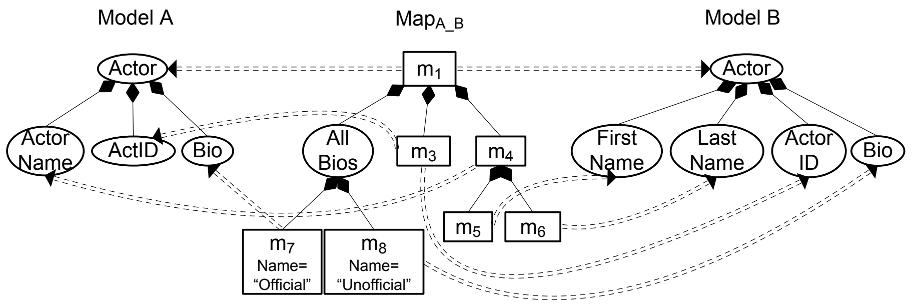


Fig. 3. A more complicated mapping between the models in Figure 1; Map_{A_B} contains an element that does not appear in either A or B

A mapping can also contain non-mapping elements that do not represent elements in either A or B but help describe how elements in A and B are related, such as $AllBios$ in Figure 3. The mapping Map_{A_B} in Figure 3 indicates that $A.Bio$ should be renamed "Official," $B.Bio$ should be renamed "Unofficial," and both are contained in a new element, $AllBios$, that appears only in Map_{A_B}.

Prior algorithms, whose mappings are not first-class models, cannot express non-equality relationships. Often, they require user intervention during $Merge$ to incorporate relationships that are more complicated than simply equating two elements. $Merge$ can encode simple correspondences in a mapping, so it can function even if a first-class mapping is unavailable.

2.3 *Merge* Semantics

The output of $Merge$ is a model that retains all non-duplicated information in A, B, and Map_{A_B}; it collapses information that Map_{A_B} declares redundant. If we consider the mapping to be a third model, this definition corresponds to the least-upper-bound defined in [4]: "a schema that presents all the information of the schemas being merged, but no additional information." We require $Merge$ to be generic in the sense that it does not require its inputs or outputs to satisfy any given meta-model. We now define the semantics of $Merge$ more precisely. The function "$Merge(A, Map_{A\_B}, B) \rightarrow G$" merges two models A and B based on a mapping Map_{A_B}, between A and B, producing a new model G that satisfies the following *Generic Merge Requirements* (GMRs):

1. **Element preservation:** Each element in the input has a corresponding element in G. Formally: each element $e \in A \cup B \cup Map_{A\_B}$ corresponds to exactly one element $e' \in G$. We define this correspondence as $\chi(e, e')$.
2. **Equality preservation:** Input elements are mapped to the same element in G if and only if they are equal in the mapping, where equality in the mapping is transitive. Formally: two elements $s, t \in$A$\cup B$ are said to be equal in Map_{A_B} if there is an element $v \in A \cup B$ and an equality mapping element x such that $Me(x, s)$ and $Me(x, v)$, where either $v = t$ or v is equal to t in Map_{A_B}. If two elements $s, t \in A \cup B$ are equal in Map_{A_B}, then there exists a unique element $e \in G$ such that $\chi(s, e)$ and $\chi(t, e)$. If not, then there is no such e, so s and t correspond to different elements in G.
3. **Relationship preservation:** Each input relationship is explicitly in or implied by G. Formally: for each relationship $R(s, t) \in A \cup B \cup Map_{A\_B}$ where $s, t \in A \cup B \cup Map_{A\_B}$ and R is not a mapping relationship $Me(s, t)$ or $Ms(s, t)$, if $\chi(s, s')$ and $\chi(t, t')$, then either $s' = t'$, $R(s', t') \in G$, or $R(s', t')$ is implied in G.
4. **Similarity preservation:** Elements that are declared to be similar (but not equal) to one another in Map_{A_B} retain their separate identity in G and are related to each other by some relationship. More formally, for each pair of elements $s, t \in A \cup B$, where s and t are the destination of similarity mapping relationships originating at a mapping element, x, in Map_{A_B} and s and t are not equal, there exist elements e, s', $t' \in G$ and a meta-model-specific non-mapping relationship R such that $\chi(s, s')$, $\chi(t, t')$, $R(e, s')$, $R(e, t')$, $\chi(x, e)$, and e includes an expression relating s and t.
5. **Meta-meta-model constraint satisfaction:** G satisfies all constraints of the meta-meta-model. G may include elements and relationships in addition to those specified above that help it satisfy these constraints. Note that we do not require G to conform to any meta-model.
6. **Extraneous item prohibition:** Other than the elements and relationships specified above, no additional elements or relationships exist in G.
7. **Property preservation:** For each element $e \in G$, e has property p if and only if $\exists t \in A \cup B \cup Map_{A\_B}$ s.t. $\chi(t, e)$ and t has property p.
8. **Value preference:** The value, v, of a property p, for an element e is denoted $p(e) = v$. For each $e \in G$, $p(e)$ is chosen from mapping elements corresponding

to e if possible, else from the preferred model if possible, else from any element that corresponds to e. More formally:

- $T = \{t \mid \chi(t, e)\}$
- $J = \{j \in (T \cap Map_{A\_B}) \mid p(j)$ is defined$\}$
- $K = \{k \in (T \cap$ the preferred model$) \mid p(k)$ is defined$\}$
- $N = \{n \in T \mid p(n)$ is defined$\}$
  - If $J \neq \oslash$ then $p(e) = p(j)$ for some $j \in J$
  - Else if $K \neq \oslash$, then $p(e) = p(k)$ for some $k \in K$
  - Else $p(e) = p(n)$ for some $n \in N$

GMR 8 illustrates our overall conflict resolution strategy: give preference first to the option specified in the mapping (i.e., the explicit user input), then to the preferred model, else choose a value from one of the input elements. The ID and $History$ properties are determined differently as discussed in Section 5. For example, the result of merging the models in Figure 3 is shown in Figure 4. Note that the relationships $Actor - FirstName$ and $Actor - LastName$ in model B and the $Actor - Bio$ relationships in both models are implied by transitivity in Figure 4, so GMR 3 is satisfied.

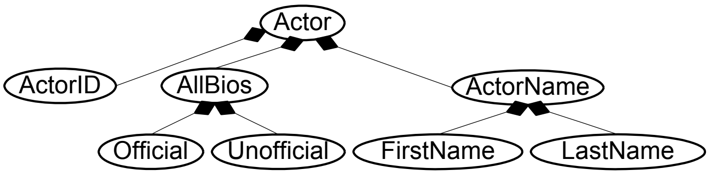


Fig. 4. The result of performing the merge in Figure 3

The GMRs are not always satisfiable. For example, if there are constraints on the cardinality of relationships that are incident to an element, then there may be no way to preserve all relationships. Depending on the relationships and meta-meta-model constraints, there may be an automatic resolution, manual resolution or no resolution that satisfies the GMRs. In Section 4 we present conflict resolutions for some common constraints and discuss when such resolution can be automatic. We also specify default resolution strategies for each category of constraint and note when resolution can be made to satisfy the GMRs.

3 Conflict Resolution

Determining the merged model requires resolving conflicts in the input. We categorize conflicts based on the meta-level at which they occur: representation conflicts (Section 3.1) occur at the model level, meta-model conflicts (Section 3.2) occur at the meta-model level, and fundamental conflicts (Section 3.3) occur at the meta-meta-model level.

3.1 Representation Conflicts

A representation conflict arises when two models describe the same concept in different ways. For example, in Figure 1 model A represents $Name$ by one element, $ActorName$, while model B represents it by two elements, $FirstName$ and $LastName$. After merging the two models, should $Name$ be represented by one, two or three elements? The decision is application dependent.

The input mapping resolves representation conflicts in $Merge$. Since the mapping is a model, it can specify that elements in models A and B are either:

– The same, by being the destination of equality mapping relationships that originate at the same mapping element. $Merge$ can collapse these elements into one element that includes all relationships incident to the elements in the conflicting representations.
– Related by relationships and elements in our meta-meta-model. E.g., we can model $FirstName$ and $LastName$ in A as child elements of $ActorName$ in B by the mapping shown in Figure 3.
– Related in some more complex fashion that we cannot represent using our meta-meta-model's relationship kinds. E.g., we can represent that $ActorName$ equals the concatenation of $FirstName$ and $LastName$ by a mapping element that has similarity mapping relationships incident to all three and an $Expression$ property describing the concatenation. Resolution can be done by a later operator that understands the semantics of $Expression$.

The mapping can also specify property values. For example, in Figure 3 Map_{A_B} specifies that the elements contained by $AllBios$ should be named $Official$ and $Unofficial$.

Solving representation conflicts has been a focus of the ontology merging literature ([7,8]) and of database schema merging ([1,11]). Since representation conflicts are resolved before $Merge$ occurs, we do not discuss their resolution.

3.2 Meta-model Conflicts

Traditionally, merge results are required to conform to a given meta-model. Since $Merge$ is meta-model independent, it does not resolve meta-model conflicts. We therefore introduce an operator, $EnforceContraints$, that coerces a model to obey a set of constraints. This operator is necessarily meta-model specific. However, it may be possible to implement it in a generic way, driven by a declarative specification of each meta-model's constraints. $EnforceContraints$ would enforce other constraints, such as integrity constraints, as well. Since meta-model conflicts are resolved after $Merge$, we do not discuss their resolution.

3.3 Fundamental Conflicts

A fundamental conflict occurs when the result of $Merge$ would not be a model due to violations of the meta-meta-model. This is unacceptable because later operators would be unable to manipulate it.

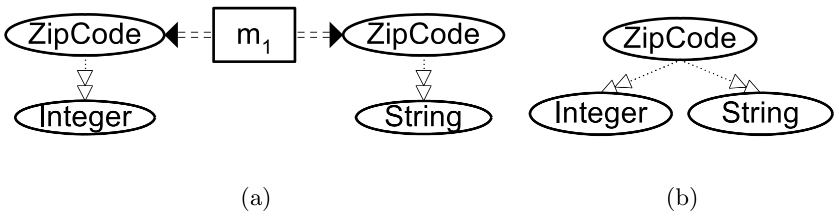


Fig. 5. A merge input (a) and its result (b) that violates the one-type restriction. The double triangle arrow represents the *Type-of* relationship.

One possible meta-meta-model constraint is that an element has at most one type. We call this the one-type restriction. Given this constraint, an element with two types manifests a fundamental conflict. For example in the model fragments in Figure 5(a) *ZipCode* has two types: *Integer* and *String*. In the *Merge* result in Figure 5(b), the two *ZipCode* elements are collapsed into one element. But the type elements remain separate, so *ZipCode* is the origin of two type relationships.

Since *Merge* must return a well-formed instance of the meta-meta-model, it must resolve fundamental conflicts. Resolution rules for some fundamental conflicts have been proposed, such as [4] for the one-type restriction. We identify other kinds of fundamental conflicts and resolution rules for them in Section 4 and incorporate them into our generic *Merge*.

4 Resolving Fundamental Conflicts

This section briefly discusses resolution of one fundamental conflict in *Merge* — the one-type conflict. Full discussion of fundamental conflicts (including what features lead to an automatic *Merge*, when manual intervention is required, and default resolutions) can be found in [9].

Many meta-meta-models restrict some kinds of relationships to a maximum or minimum number of occurrences incident to a given element. For example, the one-type restriction says that no element can be the origin of more than one *Type-of* relationship. Such restrictions can specify minima and/or maxima on origins or destinations of a relationship of a given kind. *Merge* resolves one-type conflicts using a customization of the BDK algorithm ([4]) for Vanilla, which is described in [9]. Recall Figure 5 where the merged *ZipCode* element has both *Integer* and *String* types. The BDK resolution creates a new type that inherits from both *Integer* and *String* and replaces the two *Type-of* relationships from *ZipCode* by one *Type-of* relationship to the new type, as shown in Figure 6. By the implication rules of Vanilla, both of the original relationships (*ZipCode* is of type *Integer* and *String*) are implied [9].

This approach to resolving one-type conflicts is an example of a more general approach, which is the one we use as a default: to resolve a conflict, alter explicit relationships so that they are still implied and the GMRs are still satisfied.

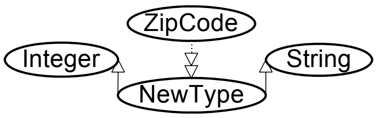


Fig. 6. Resolving the one-type conflict of Figure 5. The single triangle arrow represents the Is-A relationship.

Since the default resolution may be inadequate due to application-specific requirements, $Merge$ allows the user to either (1) specify an alternative function to apply for each conflict resolution category or (2) resolve the conflict manually.

5 The *Merge* Algorithm

The following describes an algorithm for $Merge$ that satisfies the GMRs:

1. Initialize the merge result G to $\oslash$.
2. Elements: Induce an equivalence relation by grouping the elements of A, B, and Map_{A_B}. Initially each element is in its own group. Then:
   (a) If a relationship $Me(d, e)$ exists between an element $e \in (A \cup B)$ and a mapping element $d \in Map_{A\_B}$, then combine the groups containing d and e.
   (b) Iterate (a) to a fixpoint and create a new element in G for each group.
3. Element Properties: Let e be a merged element in G corresponding to a group I. The value v of property p of e, $p(e) = v$, is defined as follows:
   (a) The properties of e are the union of the properties of the elements of I. $Merge$ determines the values of properties of e other than $History$ and ID as follows:
   $J = \{j \in (I \cap Map_{A\_B}) \mid p(j) \text{ is defined}\}$
   $K = \{k \in (I \cap \text{the preferred model}) \mid p(k) \text{ is defined}\}$
   $N = \{n \in I \mid p(n) \text{ is defined}\}$

      i. If $J \neq \oslash$, then $p(e) = p(j)$ for some $j \in J$
      ii. Else if $K \neq \oslash$, then $p(e) = p(k)$ for some $k \in K$
      iii. Else $p(e) = p(n)$ for some $n \in N$

   By definition of N, some value for each property of e must exist. In (i) – (iii) if more than one value is possible, then one is chosen arbitrarily.
   (b) Property $ID(e)$ is set to an unused ID value. Property $History(e)$ describes the last action on e. It contains the operator used (in this case, $Merge$) and the ID of each element in I. This implicitly connects the $Merge$ result to the input models and mapping without the existence of an explicit mapping between them.
4. Relationships: For every two elements e' and f' in G that correspond to distinct groups E and F, where E and F do not contain elements that are the origin of similarity mapping relationships, if there exists $e \in E$ and

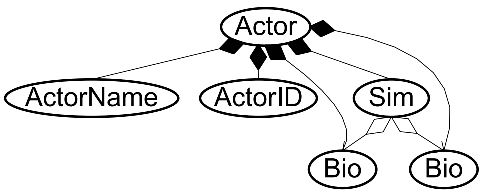


Fig. 7. Results of *Merge* on Figure 2. The empty diamond headed arrow indicate a *Has-a* relationship.

$f \in F$ such that $R(e, f)$ is of kind t and has cardinality c, then create a (single) relationship $R(e', f')$ of kind t and cardinality c. Reflexive mapping relationships (i.e., mapping relationships between elements that have been collapsed) are excluded since they no longer serve a purpose.

(a) Replace each similarity mapping relationship, Ms, whose origin is m by a weak-aggregation relationship — in Vanilla, this would be the *Has-a* relationship. This relationship has an origin of e and and a destination of the element of G that corresponds to Ms's destination's group. For example, if the two *Bio* elements in Figure 1 were connected by similarity mapping relationships instead of equality mapping relationships, the result would be as in Figure 7.

(b) Relationships originating from an element are ordered as follows:

i. First those corresponding to relationships in Map_{A_B},

ii. Then those corresponding to relationships in the preferred model but not in Map_{A_B},

iii. Then all other relationships. Within each of the above categories, relationships appear in the order they appear in the input. Finally, *Merge* removes implied relationships from G until a minimal covering remains.

5. Fundamental conflict resolution: After steps (1) – (4) above, G is a duplicate-free union of A, B, and Map_{A_B}. For each fundamental conflict in G (if any), if a special resolution strategy has been defined, then apply it. If not, apply the default resolution strategy.

Resolving one conflict may interfere with another, or even create another. If interference between conflict resolution steps is a concern in another meta-meta-model, then *Merge* can apply a priority scheme based on an ordered list of conflict resolutions. The conflict resolutions are then applied until reaching fixpoint. Since resolving one-type conflicts cannot create cycles in Vanilla, conflict resolution in Vanilla is guaranteed to terminate. However, conflict resolution rules in other meta-meta-models must be examined to avoid infinite loops.

The algorithm described above satisfies the GMRs in Section 2.3. We can see this as follows:

– Step 1 (Initialization) initializes G to the empty set.
– Step 2 (Elements) enforces GMR 1 (Element preservation). It also enforces the first direction of GMR 2 (Equality preservation); elements equated by Map_{A_B} are equated in G. No other work is performed in step 2.

- Step 3 (Element properties) performs exactly the work in GMR 7 (Property preservation) and GMR 8 (Value preference) except for the refinements in steps 3b and 3c for the ID and $History$ properties. No other work is performed in step 3.
- In step 4 (Relationships), step 4a enforces GMR 3 (Relationship preservation) and step 4b enforces that a relationship exists between elements mapped as similar, as required in GMR 4 (Similarity preservation). Step 4d removes only relationships that are considered redundant by the meta-meta-model. Step 4c (relationship ordering) is the only step not explicitly covered by a GMR, and it does not interfere with any other GMRs.
- Step 5 (Fundamental conflict resolution) enforces GMR 5 (meta-meta-model constraint satisfaction) and performs no other work.

If special resolution strategies in step 5 do nothing to violate any GMR or equate any elements not already equated, GMRs 2 (Equality preservation), 4 (Similarity preservation) and 6 (Extraneous item prohibition) are satisfied, and all GMRs are satisfied. Other than special properties (ID and $History$) and the ordering of relationships, no additional work is performed beyond what is needed to satisfy the GMRs.

6 Algebraic Properties of Merge

It is important that the result of one or more invocations of $Merge$ be insensitive to the order in which models are merged. Otherwise, it would be difficult to predict the result of using $Merge$, especially when more than two models need to be merged, since this requires invoking a sequence of merge operators.

In this section, we show that $Merge$ is well-behaved. In Section 6.1 we show that it is commutative, meaning that the order of two models being merged has no effect on the result. In Section 6.2 we show that $Merge$ is associative in the simple case of three models connected by two mappings. That is, given three models M, N, and O that are related by two mappings Map_{M_N} and Map_{N_O}, the result of merging them is the same whether M and N are merged first or N and O are merged first.

There are more general cases where the order of merges affects the choice of mappings that are used to drive each merge. In addition to Map_{M_N} and Map_{N_O}, if there is also a third mapping Map_{M_O} between models M and O, then the choice of which pair of models to merge first can affect the result. In Section 6.3 we give conditions under which $Merge$ is commutative and associative in such cases.

For ease of exposition we only consider cases where the outcome of each invocation of $Merge$ is uniquely specified by the inputs (e.g., exactly one correct choice of value exists for each property).

6.1 Commutativity

We say that $Merge$ is commutative if, for any pair of models M and N and any mapping Map_{M_N} between them, $Merge(M, Map_{M\_N}, N) = Merge(N,$

Map_{M_N}, $M) = G$. We argue that $Merge$ is commutative under two assumptions: (1) the same model is the preferred model in both invocations of $Merge$ and (2) if there are unspecified choices to be made (e.g., choosing a property value from among several possibilities, each of which is allowed by $Merge$), the same choice is made in both invocations of $Merge$. We begin by showing that commutativity holds for $Merge$ as specified by the GMRs and then show that it holds for the $Merge$ algorithm specified in Section 5.

The commutativity of $Merge$ as specified by the GMRs in Section 2.3 follows directly from their definition, since the GMRs are symmetric: Rules 1–4 and 6–7 are inherently symmetric. Rule 8 (Value preference) is symmetric as long as the preferred model is the same in both invocations of $Merge$ and unspecified choices are the same in both invocations of $Merge$, as stipulated in (2) above. Rule 5 is the resolution of fundamental conflicts. In Vanilla this is symmetric, but $Merge$ in other meta-meta-models may not be commutative, depending on their conflict resolution rules. It is worth noting that other than the mapping relationships, GMR 3 (Relationship Preservation) and $Merge$ in general treat the relationships in the same way and thus no special handling is required for the different relationships.

The algorithm specification in Section 5 is commutative as well; again we show this from the algorithm's symmetry. Steps 1 (Initialize) and 2 (Elements) are symmetric. Steps 3 (Element properties) and 4 (Relationships) are symmetric as long as the preferred model is the same in both merges and arbitrary choices are the same, as stipulated in (2) above. Step 5 (Fundamental conflict resolution) is symmetric if the conflict resolutions are symmetric. As argued above, this holds for conflict resolution in Vanilla, and hence the $Merge$ algorithm is symmetric and thus commutative in Vanilla.

6.2 Associativity

We say that two models are isomorphic if there is a 1:1 onto correspondence between their elements, and they have the same relationships and properties (but the values of their properties and the ordering of their relationships may differ). $Merge$ is associative if, for any three models M, N, and O, and any two mappings Map_{M_N} (between M and N) and Map_{N_O} (between N and O), R is isomorphic to S where:

$$\begin{aligned}
&P = Merge(M, Map_{M\_N}, N)\\
&Q = Merge(N, Map_{N\_O}, O)\\
&Map_{P\_N} = Match(P, N)\\
&Map_{N\_Q} = Match(N, Q)\\
&R = Merge(P, Compose(Map_{P\_N}, Map_{N\_O}), O)\\
&S = Merge(M, Compose(Map_{M\_N}, Map_{N\_Q}), Q)
\end{aligned}$$

$Match(P, N)$ and $Match(N, Q)$ compute χ as defined in the GMRs by matching the IDs of N with the IDs in the $History$ property of P and Q respectively. N, P, Q, $Match(P, N)$, and $Match(N, Q)$ are shown in Figure 8.

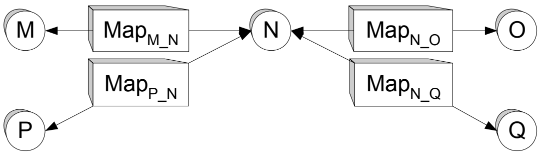


Fig. 8. Showing associativity requires intermediate mappings

The *Compose* operator takes a mapping between models A and B and a mapping between models B and C and returns the composed mapping between A and C. Consider $Compose(Map_{P\_N}, Map_{N\_O})$. Intuitively it must transfer each mapping relationship of Map_{N_O} having a destination in N to a relationship having a destination in P. Since Map_{P_N} maps each element in N to exactly one element in P, any *Compose* operator will provide this functionality (such as the one described in [9]). $Compose(Map_{M\_N}, Map_{N\_Q})$ operates similarly.

A morphism is a set of directed morphism relationships from elements of one model to elements of another. To show that the two final merged models R and S are isomorphic, we define a morphism $\varphi(R \to S)$ and show that (i) φ is 1:1 and onto, (ii) $R(x, y) \in R_R$ if and only if $R(\varphi(x), \varphi(y)) \in R_S$, and (iii) x has property p if and only if $\varphi(x)$ has property p. We initially consider the result of *Merge* ignoring the fundamental conflict resolution. We phrase the argument in terms of the GMRs. The end of Section 5 shows that the algorithm maintains all of the GMRs. The only additional work done by the merge algorithm beyond the GMRs is (1) to order the relationships and (2) set the value of the ID property. Since the latter two additions do not affect the isomorphism, so we do not repeat the associativity argument for the algorithm.

We create φ as follows. First we create the morphisms shown as arrows in Figure 9 by using *Match* and *Compose* the same way they were used to create R and S. We refer to the morphisms in Figure 9 that start at R as $Morph_R$ and those that end at S as $Morph_S$. Next we create five morphisms from R to S by composing $Morph_R$ with $Morph_S$. φ is the duplicate-free union of these five morphisms.

We want to show that φ is an isomorphism from R to S; this will show that R and S are isomorphic to one another and hence that *Merge* is associative. We first show that φ is onto (i.e., for all $y \in S$, there exists $x \in R$ such that $\varphi(x) = y$):

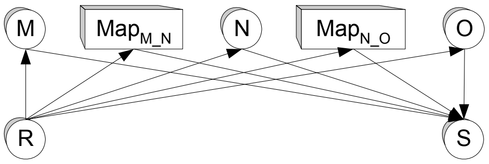


Fig. 9. Initial morphisms created to show associativity

1. Let T be the set of elements in M, Map_{M_N}, N, Map_{N_O}, and O.
2. By GMRs 1 (Element preservation) and 2 (Equality preservation) and the definitions of $Match$ and $Compose$, each element in T is the destination of exactly one morphism relationship in $Morph_R$. I.e., each element in M, Map_{M_N}, N, Map_{N_O}, and O corresponds to exactly one element in the merged model. By GMR 6 (Extraneous item prohibition) and the definitions of $Match$ and $Compose$ each element in R is the origin of at least one morphism relationship to T. I.e., each element in R must correspond to some element in M, Map_{M_N}, N, Map_{N_O}, or O. We are not considering conflict resolution, so no elements are introduced due to GMR 5 (Meta-meta-model constraint satisfaction).
3. Similarly each element of T is the origin of exactly one morphism relationship in $Morphs_S$ and each element in S is the destination of at least one $Morphs_S$ morphism relationship from T.
4. Hence by steps 2 and 3 and the definitions of $Match$ and $Compose$, φ is onto.

Next we show that φ is 1:1(i.e., $\forall x_1, x_2 \in R, \varphi(x_1) = \varphi(x_2) \rightarrow x_1 = x_2$). The proof is by contradiction.

1. Suppose φ is not 1:1. Then there must exist some x_1 $x_2 \in R$ s.t. $x_1 \neq x_2$ and $\varphi(x_1) = \varphi(x_2)$. Let $\varphi(x_1) = \varphi(x_2) = s$.
2. In that case, from the definition of φ, there exists some elements $m_1, m_2 \in M \cup Map_{M\_N} \cup N \cup Map_{N\_O} \cup O$ s.t. $Morph_R(x_1, m_1)$, $Morph_R(x_2, m_2)$, $Morphs_S(m_1, s)$, $Morphs_S(m_2, s)$ where $m_1 \neq m_2$.
3. From GMR 2 (Equality preservation) and the definitions of $Match$ and $Compose$, s must be the result of merging some elements from T that were equal in some mapping. However the equating of elements is associative; this follows directly from the grouping strategy in $Merge$ step 2 (Element properties). Therefore either:
   (a) m_1 and m_2 are not equated by a mapping, then they will not be merged into the same object — in which case it cannot be that $Morphs_S(m_1, s)$, $Morphs_S(m_2, s)$, since $m_1 \neq m_2$ or
   (b) m_1 and m_2 are equated by a mapping, in which case they will be merged into the same object — in which case it cannot be that $Morph_R(x_1, m_1)$, $Morph_R(x_2, m_2)$ since $m_1 \neq m_2$.
4. Hence the equating of the elements is associative and φ is 1:1.

The next step in showing φ is an isomorphism from R to S is to show that $R(x, y) \in R_R$ if and only if $R(\varphi\ (x), \varphi\ (y)) \in R_S$. That is, a relationship exists in R if and only if a corresponding relationship exists in S. GMR 3 (Relationship preservation) guarantees that each relationship R input to $Merge$ has a corresponding relationship R' in the merged model unless R's origin and destinations collapse into one element. Similarly the $Match$ and $Compose$ definitions preserve the elements, and a relationship $R(x, y) \in R_R$ if and only if $R(\varphi(x), \varphi(y)) \in R_S$.

The last step to show that φ is an isomorphism is to show that each element $r \in R$ has property p if and only if $\varphi(r)$ has property p. GMR 7 (Property

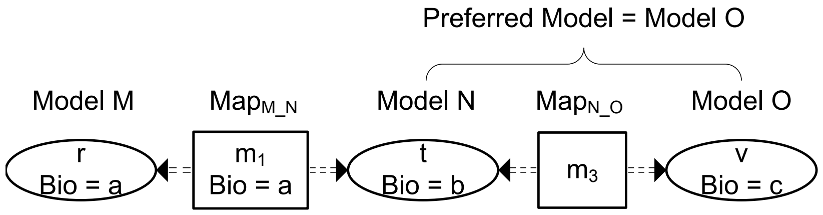


Fig. 10. A series of mappings and models

preservation) implies that each element in the merged model has a property p if and only if some input element that it corresponds to has property p. From the argument showing that φ is 1:1, we know that the equating of elements is associative, and hence $r \in R$ has property p if and only if $\varphi(r)$ has property p. Hence φ is an isomorphism from R to S and $Merge$ is associative.

$Merge$ is not associative with respect to the values of properties. Their value is determined by GMR 8 (Value preference). After a sequence of $Merge$s, the final value of a property may depend on the order in which the $Merge$s are executed, because the value assigned by the last $Merge$ can overwrite the values of any Merges that preceded it. For example, in Figure 10 the mapping element m_1 in Map_{M_N} specifies the value a for property Bio. In addition, the $Merge$ definition specifies that O is the preferred model for the merge of N and O. If the sequence of operators is:

$Merge(M, Map_{M\_N}, N) \rightarrow P$
$Merge(P, Compose(Match(P, N), Map_{N\_O}), O) \rightarrow R,$

then in model P the Bio property as a result of merging r and t will have the value a since it is specified in $m1$. In the second $Merge$, model O will be the preferred model, and the value of the Bio property of the resulting element is c.

However, if the sequence of operators is:

$Merge(N, Map_{N\_O}, O) \rightarrow Q$
$Merge(M, Compose(Map_{M\_N}, Match(N, Q)), Q) \rightarrow S,$

then the Bio property of the element that corresponds to t and v will have value c since O is the preferred model. Since the value of Bio in mapping element $m1$ is a, its value in the final result is a instead of c as in the first example.

Unless $Merge$ can express a total preference between models – which is impractical – it will not be associative with respect to the final values of properties.

Hence, ignoring conflict resolution, $Merge$ is associative. Since all of the fundamental conflict resolution in Vanilla is associative, $Merge$ is associative for Vanilla as well (see [4] for references on the associativity of the BDK).

6.3 Mapping-Independent Commutativity and Associativity

We say that $Merge$ is mapping-independent commutative (respectively associative) if it is commutative (respectively associative) even when the order of

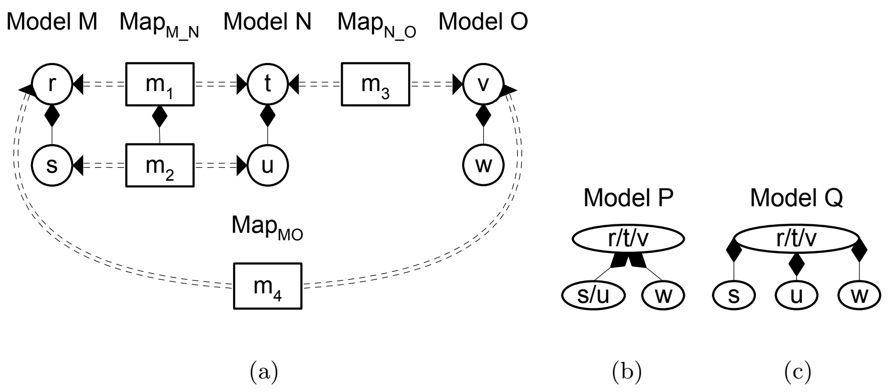


Fig. 11. A series of merges (a) A set of models and mappings. (b) the result of merging the models using Map_{M_N} and Map_{N_O}. (c) the results of merging the models using Map_{N_O} and Map_{M_O}.

Merge operations affects the choice of mapping that is used in each *Merge*. For example, consider the models and mappings in Figure 11. In (a), Map_{M_N} is the only mapping that equates elements s and u. When Map_{M_N} is used, as in (b), elements s and u are combined. However, when Map_{M_N} is not used, as in (c), s and u remain as separate elements.

When is *Merge* guaranteed to be mapping-independent associative and commutative? Ignoring meta-meta-model constraint satisfaction, given a set of models, S (e.g., M, N, O in Figure 11), and two sets of mappings $Mappings_A$ (e.g., Map_{M_N}, Map_{N_O}) and $Mappings_B$ (e.g., Map_{N_O}, Map_{M_O}) over S, in order for *Merge* to produce isomorphic results it must be the case that:

- Elements r and v are equated to one another either directly or transitively in $Mappings_A$ if and only if they are equated to one another directly or transitively in $Mappings_B$; r can be declared equal to t and t equal to v in one set of mappings and in another set of mappings r can be declared equal to v and v equal to t.
- Elements r and v are declared to be "similar to" another element in $Mappings_A$ if and only if they are declared to be "similar to" the same element in $Mappings_B$.
- Additional elements and relationships are introduced in $Mappings_A$ if and only if corresponding elements and relationships are introduced in $Mappings_B$.

Informally, we know that these are the only requirements because:

- *Merge* is associative and commutative if the mappings are the same, as shown above.
- Mappings have three roles with respect to Merge; they can (1) declare elements to be equal, (2) declare elements to be similar or (3) add in additional elements and relationships. We address each of these three roles below:

1. Equality: Since equality is transitive, we only need to enforce that elements that are equated in one set of mappings are also equated in the other.
2. Similarity: $Mappings_A$ and $Mappings_B$ must both declare that the same elements are similar if they are to be isomorphic to each other. However, similarity is not transitive. If it is used then there is an implicit restriction on the sets of mappings; if $Mappings_A$ declares an element in model $S1$ to be similar to an element in model $S2$, then $Mappings_B$ must contain a mapping between $S1$ and $S2$ in order for the similarity relationship to be expressed. We do not need to consider the more complicated case when one mapping declares two elements to be similar through a mapping element, s, and then another mapping element, t, declares s to be similar to some other element because by our problem definition the set of mappings cannot map results of previous $Merge$s.
3. Additional elements and relationships: Finally, because mappings can also add elements and relationships, if $Mappings_A$ adds an element or a relationship, then $Mappings_B$ must add a corresponding element or relationship, too. However, as with similarity, there may be an implicit restriction on the set of mappings; if $Mappings_A$ declares an element in model $S1$ to contain an element in model $S2$, then $Mappings_B$ must contain a mapping between $S1$ and $S2$ for the $Contains$ relationship to be expressed. Again, because the set of mappings cannot map results of previous merges, we need not consider more complicated cases.

7 Conclusions

The problem of merging models lies at the core of many meta data applications. In each case, two given models need to be combined into one. Because there are many different types of models and applications, this problem has been tackled independently in specific domains many times. In [9] we generalized that work by presenting a generic operator, $Merge$, that can be applied in all of these domains. This paper extended [9] by showing that generic $Merge$ is commutative and associative.

References

1. Batini, C., Lenzerini, M., Navathe, S.B.: A comparative analysis of methodologies for database schema integration. ACM Computing Surveys 18(4), 323–364 (1986)
2. Beeri, C., Milo, T.: Schemas for integration and translation of structured and semi-structured data. In: ICDT, pp. 296–313 (1999)
3. Bergamaschi, S., Castano, S., Vincini, M.: Semantic integration of semistructured and structured data sources. SIGMOD Record 28(1), 54–59 (1999)
4. Buneman, P., Davidson, S.B., Kosky, A.: Theoretical aspects of schema merging. In: Pirotte, A., Delobel, C., Gottlob, G. (eds.) EDBT 1992. LNCS, vol. 580, pp. 152–167. Springer, Heidelberg (1992)

5. Calvanese, D., De Giacomo, G., Lenzerini, M.: What can knowledge representation do for semi-structured data? In: AAAI (1998)
6. Doan, A., Halevy, A.: Semantic integration research in the database community: A brief survey. AI Magazine 26(1), 83–94 (2005)
7. Noy, N.F., Musen, M.A.: Smart: Automated support for ontology merging and alignment. In: Banff Workshop on Knowledge Acquisition, Modeling, and Management (1999)
8. Noy, N.F., Musen, M.A.: Prompt: Algorithm and tool for ontology merging and alignment. In: AAAI (2000)
9. Pottinger, R.A., Bernstein, P.A.: Merging models based on given correspondences. In: VLDB, pp. 862–873 (2003)
10. Rahm, E., Bernstein, P.A.: A survey of approaches to automatic schema matching. VLDB Journal 10(4), 334–350 (2001)
11. Spaccapietra, S., Parent, C.: View integration: A step forward in solving structural conflicts. TKDE 6(2), 258–274 (1994)

The History of WebML
Lessons Learned from 10 Years of Model-Driven Development of Web Applications

Stefano Ceri, Marco Brambilla, and Piero Fraternali

Dipartimento di Elettronica e Informazione, Politecnico di Milano
Piazza L. Da Vinci, 32. I20133 Milano, Italy
{ceri,mbrambil,fraterna}@elet.polimi.it

Abstract. This work presents a retrospective analysis on the conceptual modeling language for Web applications called WebML, which was first defined about 10 years ago. WebML has been an incubator for research on conceptual modeling, exploiting existing experiences in the field and continuously addressing new challenges concerning abstractions, methods, tools, and technologies. People working on WebML are spread among universities, technology transfer centres, and a spin-off. In this paper, we illustrate the history of WebML, we summarize the essence of the approach, and we sketch the main research branches that spawned from the initial proposal. We describe how new trends in research, application development, methodology, and tool prototyping led to the continuous growth of the modeling language.

1 Introduction

Data-intensive Web applications, i.e., software systems whose main purpose is to give access to well-organized content, represented the first industrial application of the Web, and are still predominant in terms of volume and commercial value. All companies have an institutional site showing their business and describing their offers, and many enterprises manage the relations with their customers through the Web. Therefore, these applications have been a preferred target of development methods and tools, which have been available for a long time.

Among them, the Web Modelling Language (WebML) [1] was defined, about 10 years ago, as a conceptual model for data-intensive Web applications. In the early days of Web development, technologies were immature and in perpetual change; as a reaction, WebML was conceived as a high level, implementation-independent conceptual model, and the associated design support environment, called WebRatio [9], has always been platform-independent, so as to adapt to frequent technological changes.

While other conceptual models focus more on the early phases of the development process (i.e., requirement specification [21]), WebML concentrates on the later phases, starting from design, down to the implementation. As many other conceptual modeling languages [14], WebML is based upon the principle of separation of concerns: content, interface logics, and presentation logics are defined as separate models of the application. The main innovation in WebML comes from the hypertext modelling notation (patented in 2003), which enables the specification of Web pages consisting of conceptual components (units) interconnected by conceptual links. The hypertext model is drawn in a simple and quite intuitive visual notation, but has a rigorous semantics, which allows the automatic transformation of diagrams into the complete running code of a data-intensive Web application. Originally, the focus of the design of WebML concentrated on the definition of a powerful set of units; with time, we realized that units are just specific components, which can be defined and adapted to the needs of any new technological development; instead, the essence of the WebML hypertext model lies in the rules for assembling components and links into a graph, and for inferring all the possible parameter passing rules from the component interfaces and the link types. A well-formed graph guarantees the correct data flow among units and dictates the proper component execution order when computing the content of pages. Ultimately, computing a hypertext model amounts to enacting a workflow of component execution driven by the user's "clicking behaviour". In retrospective, the choices of link and component semantics were quite adequate to the purpose and remained stable throughout ten years of language evolution.

While the Web has gone through waves of innovation, new technological scenarios have developed, and revolutionary concepts – such as enabling the interaction of software programs rather than only humans – have emerged. Several new challenges have been addressed within the WebML context, including:

- Web services and service-oriented architectures [11];
- Integration with business processes [5];
- Personalization, adaptation, context awareness, and mobility [6];
- Semantic Web and Semantic Web Services [4];
- Rich Internet Applications [3];
- Search-based applications;
- Support of reuse, multi-threading, and modularization.

This paper highlights the core nucleus of the WebML language, which has remained stable over the years, and illustrates how we have dealt with each new challenge through a four-step approach. The treatment of each extension is necessarily concise and visual, for more details we refer readers to published papers and reports.

2 The Original WebML Language

The specification of a Web application in WebML [2] consists of **a set of orthogonal models**: the application *data model* (an extended Entity-Relationship model), one or more *hypertext models* (i.e., different site views for different types of users), expressing the navigation paths and the page composition; and the *presentation model*, describing the visual aspect of the pages. We focus on the hypertext model, as the data model is not innovative; the presentation model is also quite interesting, as it enables "dressing" a hypertext model to obtain Web pages with the desired layout and look&feel for any rendition technology, but is also outside the scope of this paper.

2.1 The WebML Hypertext Model

A hypertext model consists of one or more *site views*, each of them targeted to a specific user role or client device. A site view is a collection of pages (possibly grouped into *areas* for modularization purposes); the content of pages is expressed by components for data publishing (called *content units*); the business logic triggered by the user's interaction is instead represented by sequences of *operation units*, which denote components for modifying data or for performing arbitrary business actions (e.g., sending email). Content and operations units are connected by *links*, which specify the data flow between them and the process flow for computing page content and for enacting the business logic, in reaction to user's generated navigation events.

Consider for instance a simple scenario: users browse a Home Page, from where they can navigate to a page showing an index of loan products. After choosing one loan, users are lead to a page with the loan details and the list of proposals for the chosen loan. The WebML specification for the described hypertext is depicted in Figure 1. The Home Page contains only some static content, which is not modeled. A link from this page leads to the Loans page, containing an index of all loans, graphically represented by means of an *index unit* labeled Loans Index. When the user selects a loan from the index, he is taken to the Chosen Loan page, showing the loan details. In this page, a *data unit*, labeled Loan Details, displays the attributes of the loan (e.g. the company, the total amount and the rate), and is linked to another index unit, labeled Proposals Index, which displays the plan options.

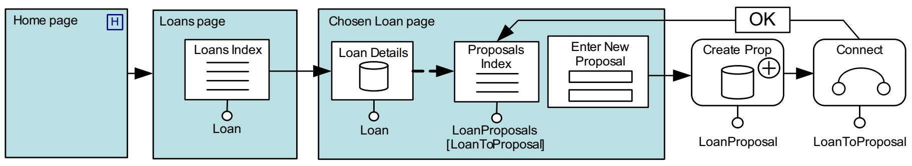


Fig. 1. A WebML hypertext for browsing and updating information

This example contains **units for publishing content** (data and index units), which display some of the attributes of one or more instances of a given entity. Syntactically, each type of unit has a distinguished icon and the entity name is specified at the bottom of the unit; below the entity name, predicates (called *selectors*) express conditions filtering the entity instances to be shown. The example of Figure 1 also shows static content units, which display fixed content not coming from the objects in the data model: this is the case of the Enter New Proposal *entry unit*, which denotes a form for data entry. The hypertext model also illustrates the use of operation units: the outgoing link of the Enter New Proposal entry unit activates a sequence of two operation units: a create and a connect unit, which respectively create an instance of the LoanProposal entity and connect it with a relationship instance to the Loan entity.

WebML distinguishes between normal, transport, and automatic links. **Normal links** (denoted by solid arrows) enable navigation and are rendered as hypertext anchors or form buttons, while **transport links** (denoted by dashed arrows) enable only parameter passing and are not rendered as navigable widgets. **Automatic links** (denoted by an [A] icon) are normal links, which are automatically "navigated" by the system on page load. Orthogonally, links can be classified as contextual or non-contextual: **contextual links** transfer data between units, whereas **non-contextual links** enable navigation between pages, with no associated parameters. Operation units also demand two other types of links: **OK links** and **KO links**, respectively denoting the course of action taken after success or failure in the execution of the operation. In the example of Figure 1:

- The link from the Home page to the Loans page is non-contextual, since it carries no information, and simply enables a change of page.
- The link from the Loans Index unit to the Loan Details unit is normal and contextual, as it transports the ID of the loan chosen in the index unit and displayed in the data unit.
- The link from the Loan Details data unit to the Proposals Index unit is a transport link: when the user enters the Chosen Loan page, the Loan Details unit is displayed and, at the same time, the Loan ID is transferred to the Proposal Index unit, so that the content of the Proposals index unit is computed and displayed without user's intervention. No navigable anchor is rendered, because there is no need of the user's interaction.
- The outgoing link of the Connect unit, labelled OK, denotes that after the successful execution of the operation the Choose Loan page is displayed.

The content of a unit depends on its input links and local selectors. For instance, the Loan ID is used to select those proposals associated with a given loan by the relationship role LoanToProposal; this selection is expressed by the selector condition [LoanToProposal] below the unit's entity. In general, arbitrary logical conditions can be used, but conjunctive expressions are easily presented in the diagrams, where each conjunct is a predicate over an entity's attribute or relationship role.

2.2 Semantics of the WebML Hypertext Model

As already mentioned, WebML is associated with a page computation algorithm deriving from the formal definition of the model's semantics (based on statecharts in [10]). The essential point is the **page computation algorithm**, which describes how the content of the page is determined after a navigation event produced by the user. Page computation amounts to the progressive evaluation of the various units of a page, starting from input parameters associated with the navigation of a link. This process implies the orderly propagation of the value of link parameters, from an initial set of units, whose content is computable when the page is accessed, to other units, which expect input from automatic or transport links exiting from the already computed units of the page.

In WebML, pages are the fundamental unit of computation. A WebML page may contain multiple units linked to each other to form a complex graph, and may be accessed by means of several different links, originating from other pages, from a unit inside the page itself, or from an operation activated from the same page or from another page. The content of a page must be computed or recomputed in the following cases:

- When the page is entered through a link (contextual or non-contextual) originating in another page; in this case the contents of all units of the page are calculated afresh, based on the possible parameter values carried by the link.
- When the user navigates an intra-page link and thus supplies some new input to the destination unit of the link; in this case, part of the content of the page is calculated based on the parameter values associated with the intra-page link, but part of the content of the page is computed based on the values of parameters existing prior to the navigation of the intra-page link, so that past user's choices are not lost when navigating the link.
- When an operation is invoked, ending with a link pointing back to the same page: this case is similar to the navigation of an intra-page link, but in addition the operation may have side effects on the content visualized in the page, which may change the content displayed by the page.

The example in Figure 2 illustrates the three cases:

- When the ArtistIndex page is accessed through the non-contextual link labeled Link1 or the Artist page is accessed through the contextual link labeled Link2, the content of the entire destination page is computed afresh, taking into account the possible input values associated with the navigated link (e.g., the OID of the selected artist when Link 2 is navigated).
- When the user selects a new album from the AlbumIndex unit, new context information flows along the link labeled Link3 and determines the album to be displayed in the AlbumData unit; at the same time, the Artist displayed in the ArtistData data unit must be "remembered" and redisplayed, because the input of the ArtistData unit is not directly affected by the navigation of the intra-page link.
- When the delete operation is performed successfully and the page is re-entered through the OK link Link4, the content of the ArtistData unit is preserved, so to remember the past user's choice, whereas the content of the AlbumIndex unit and of the AlbumData unit is refreshed, so that the deleted album no longer appears in the AlbumIndex unit and in the AlbumData unit. If the delete operation fails, the KO link Link5 is followed and the content of the AlbumData unit is refreshed using the OID of the object that could not be deleted, and the content of the other units is restored. This ensures that the previously selected artist, his/her albums, and the details of the album tentatively deleted continue to be displayed when the page is re-accessed after the failed operation.

The page computation process is triggered by any of the previously discussed navigational events (inter-page link navigation, intra-page link navigation, operation activation). Based on the navigated link, a set of parameter values is collected and passed in input to the page, which determines the initial input for some of the page units. The page computation algorithm starts by tagging as computable all context-free units (e.g., units with no input parameters, like the ArtistIndex unit) and possibly

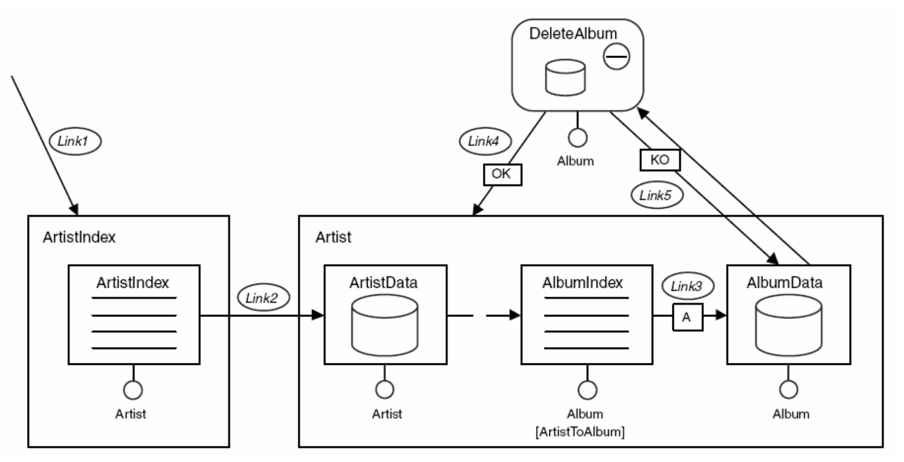


Fig. 2. Example of WebML Page with different access paths

the externally dependent units for which there are sufficient input values in the parameters passed to the page (e.g., the ArtistData unit when Link2 is navigated). Then, the computation proceeds by evaluating the units one after another, until all possible units have been evaluated. The computation process exploits the propagation of context along automatic and transport links, and a **specificity rule** telling which alternative input should be considered when multiple choices are available for evaluating the same unit.

The specificity rule introduces a partial order in parameter passing, therefore a page computation is nondeterministic (but the WebRatio tool identifies such situations and prompt designers to change the model to eliminate non-determinism). Moreover, some hypertexts can be non-computable, due to circular dependencies among units or pages, causing a deadlock in the assignment of input values to some units. The complete description of the WebML hypertext model semantics is in Chapter 5 of [8] and in [10].

2.3 The WebML Design Process

The WebML methodology exploits **a formal design process**, explained in Chapter 6 of [8], shown in Figure 3. The process includes the classic phases of requirement analysis (thoroughly addressed by dedicated methods, such as [21]), data design, hypertext design, and presentation design, followed by architecture design and implementation. The 4-step procedure, from requirements to data, to hypertext, to presentation design, can be iterated multiple times with the support of WebRatio, which acts as a rapid prototyping tool; experience has shown that a crucial advantage of using the model-driven approach comes from the ability to generate incremental prototypes directly under the eyes of the application stakeholders. Web-specific data design guidelines, based on the notion of the **web mart** as a standard ER schema that is recurrent in Web applications, have also been proposed in [9].

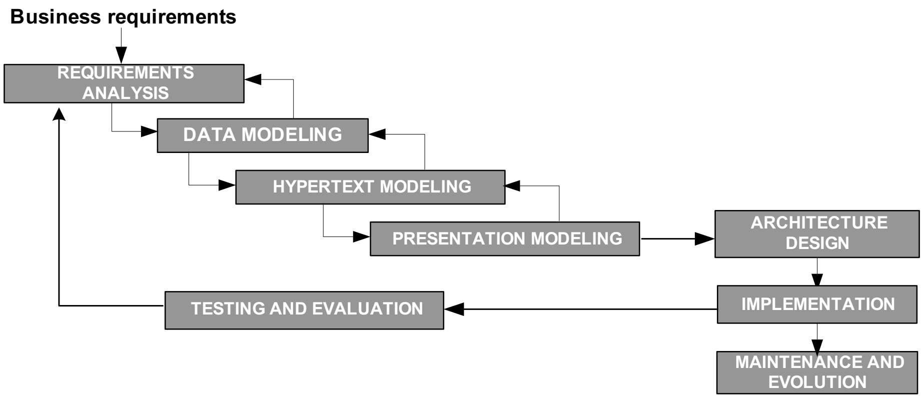


Fig. 3. The WebML Development process

2.4 The Added Value of WebML

Before describing the extensions of the original WebML model, we wish to distill the concepts that proved most valuable in the ten-year experience of development and research with WebML. Our experience demonstrated that the true added value of WebML stands in the following aspects:

- The choice of **component-based design** as the fundamental development paradigm and the **standardization of component specification**, which allow extending the language without altering its semantics.
- The use of **links of different types**, for specifying the "wiring" of components, which can be assembled into pages and sequences of operations.
- Powerful and automatic **inference rules for parameter matching**, allowing the designer to avoid explicit specification of the data carried by contextual links in all cases in which they can be deduced from the context.

The result of these design principles is an easy-to-learn formalism: the hypertext model consists of very few concepts (e.g., compared to UML): site views, areas, pages, content units, operation units, and links. At the same time, the openness of the implementation allows developers to enrich WebRatio with the components of their choice, so to achieve an almost unlimited variety of effects.

The subsequent evolution of the model built upon these aspects, adding domain-specific features to the core nucleus of the language. In retrospective, we have addressed every new challenge by using a common approach, which indeed has become evident to us during the course of time, and now is well understood and constitutes the base for all new additions. For every new research direction, four different kinds of extensions are designed, respectively addressing the development process, the content model, the hypertext meta-model, and the tool framework:

- *Extensions of the development process* capture the new steps of the design that are needed to address the new direction, providing as well the methodological guidelines and best practices for helping designers.
- *Extensions of the content model* express domain-specific standard data schemas, e.g., collections of entities and relationships, that characterize the applications in

the area of interest; the standard schema is connected with the application data model, to enable an integrated use of domain objects and special-purpose data.

- *Extension of the hypertext meta-model* refine the standard WebML concepts to capture the new abstractions required for addressing the new modelling perspective; in this way, the core semantics of WebML is preserved, but functionality is added by plugging in "libraries" of specialized concepts.
- *Extensions of the tool framework* introduce new modules in the open architecture of WebRatio (specialized data elements, new content and operation units, novel containers of model elements, etc.), implement wizards for expressing the semantics of new components in terms of existing ones, and provide code generation and runtime support for each new addition.

3 Service-Oriented Architectures

The first WebML extension discussed in this paper is towards the Service Oriented Architectures [11]. Our extension includes four components:

- The extension to the development process and the definition of some methodological guidelines for SOA design;
- A standard data schema for representing the services and the business processes to be performed;
- New WebML units for covering Web service specification and invocation, together with primitives for enforcing business process constraints;
- The support of the specified solutions through a process modeller, a translator of processes into hypertext skeletons, and an XML-to-XML mapping tool.

The extension of the **development process** to SOA requires separating application design from service design; the former addresses the front-end of a Web integration application targeted to the user, while the latter focuses on provisioning well-designed services, usable across different Web applications.

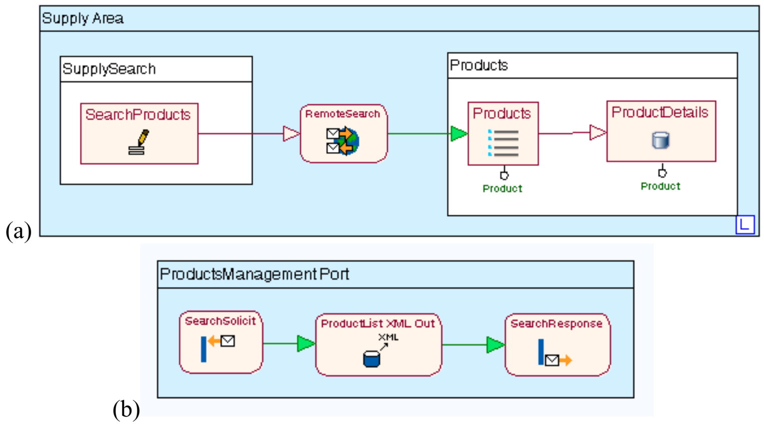


Fig. 4. Example of WebML hypertext model with invocation of remote service

The **contet model** for SOAs (not shown here for sake of brevity) supports the description of Web Services according to WSDL, including the notions of services, ports, and input/output messages.

Extensions to the **hypertext model** cover both Web Service publication and Web Service consumption. **Web Service publication** is expressed as a novel container (called *Service View*), which is analogous to a site view, but contains specifications of services instead of pages. A service specification is denoted by a *Port*, which is a container of the operations triggered upon invocation of the service.

Service invocation and **reaction to messages** are supported by specialized components, called **Web Service units**. These primitives correspond to the WSDL classes of Web service operations and comprise:

- **Web service publication primitives**: *Solicit unit* (representing the end-point of a Web service), and *Response unit* (providing the response at the end of a Web service implementation); they are used in a service view as part of the specification of the computation performed by a Web Service.
- **Web Service invocation primitives**: *Request-response* and *Request* units; they are used in site views, and denote the invocation of remote Web Services from the front-end of a web application.

For instance, Figure 4 shows a hypertext that specifies a front-end for invoking a web Service (Figure 4a) and the specification of the web Service within a port container (Figure 4b).

In the *Supply Area* of Figure 4a, the user can access the *SupplySearch* page, in which the *SearchProducts* entry unit enables the input of search keywords. The submission of the form, denoted by the navigation of the outgoing link of the entry unit, triggers a request-response operation (*RemoteSearch)*, which builds the XML input requested by the service and collects the XML response returned by it. From the service response, a set of instances of the *Product* entity are created, and displayed to the user by means of the *Products* index unit in the *Products* page; the user may continue browsing, e.g., by choosing one of the displayed products and looking at its details.

Figure 4b represents the service view that publishes the *RemoteSearch* service invoked by the previously described hypertext. The *ProductManagementPort* contains the chain of operations that make up the service: the sequence starts with the *SearchSolicit* unit, which denotes the reception of the message. Upon the arrival of the message, an XML-out operation extracts from the service provider's database the list of desired products and formats it as an XML document. The service terminates with the *SearchResponse* unit, which returns the response message to the invoker[1].

For supporting service design, WebRatio has been extended with:

- The novel Web service units.
- The *Service view* and *Port* containers.

[1] Service ports are an example of a WebML concept that has nothing to do with the user's interaction, which shows how the original target of the model (hypertext navigation) has been generalized to cover new requirements. Even more radical shifts will be needed to deal with semantic Web Services, as illustrated in the sequel.

- The runtime and code generator features necessary to produce the actual executable code corresponding to the additional modeling primitives.

4 Workflow-Driven Applications for the Web

The Web has become a popular implementation platform for B2B applications, whose goal is not only the navigation of content, but also the enactment of intra- and inter-organization business processes. Web-based B2B applications exhibit much more sophisticated interaction patterns than traditional Web applications: they back a structured process, consisting of activities governed by execution constraints, serving different user roles, whose joint work must be coordinated. They may be distributed across different processor nodes, due to organizational constraints, design opportunity, or existence of legacy systems to be reused. WebML has been extended to cover the requirements of this class of applications [5], by:

- The integration in the development life-cycle of workflow-specific deign guidelines;
- Two different models for representing the business processes;
- New design primitives (namely, WebML units) for enforcing business process constraints;
- New tools for workflow-driven application design: a process modeller and a translator of processes into hypertext skeletons.

The incorporation of business processes in WebML has been pursued in two distinct, yet complementary, scenarios:

1) S*tatic business process*, i.e., processes defined once during the design phase and then preserved for the entire application lifetime.
2) D*ynamic business processes*, in which the process schema is subject to continuous evolution.

Some aspects are common to the two scenarios, while others differ significantly. In the following, we will highlight the differences, when needed.

The WebML design process is extended with a new phase, **business process modeling**, preceding data and hypertext design. In case of *dynamic BP*, changes to the process are allowed at runtime too, and the system automatically adapts its behavior.

A **content model for static processes,** shown in Figure 5, represents meta-data about the business processes. A process is represented by the *Process* entity, associated with the *ActivityType* entity, representing the kinds of activities that can be executed in the process. An instance of a process is modeled by the *Case* entity, related to its *Process* (via the *InstanceOf* relationship) and to its activities (via the *PartOf* relationship); entity *ActivityInstance* denotes the actual instances of activities within cases.

With **dynamic processes**, the basic content model is completed by a few additional entities and relationships that represent the sequence constraints between activities, the branching and joining points, and the execution conditions. Thanks to the more refined meta-data, the application can infer the process structure and execution status at runtime and adapt to dynamic changes of the process schema.

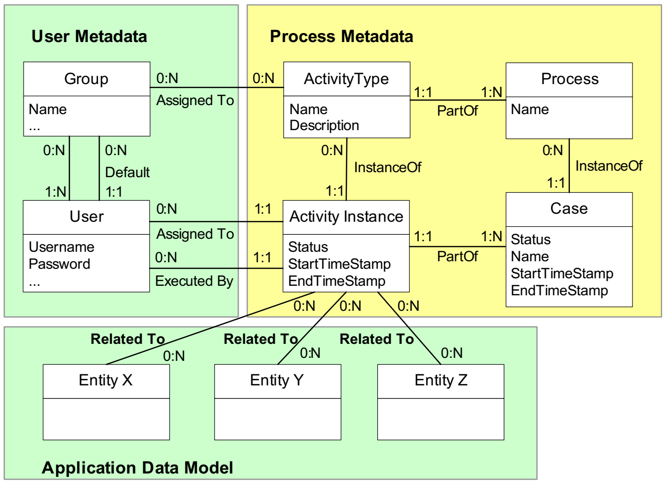


Fig. 5. Content model for the specification of a business process

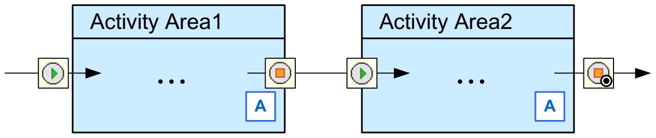


Fig. 6. Two activity areas and the *start* and *end* links that denote the initiation and termination of an activity

In case of static processes, the process structure is embodied in the topology of the hypertext. The intuition is that the process progresses as the actors navigate the front-end, provided that the hypertext model and the process metadata are kept in synch. To this end, new primitives are added to the hypertext model, for specifying activity boundaries (namely *activity areas*) and process-dependent navigation (namely *workflow links*). Figure 6 shows some of these primitives: "Activity Areas" denote groups of pages that implement the front-end for executing an activity; specialized links represent the workflow-related side effects of navigation: starting, ending, suspending, and resuming activities. Distributed processes deployed on *SOAs* can be obtained by combining the workflow and Web Services primitives [5].

For *dynamic business processes*, the next activity to be executed is not statically specified by an activity area, but is determined at runtime by a unit (called *Next unit*), which encapsulates the process control logic. It exploits the information stored in the process meta-data and log to calculate the current process status and the enabled state transitions. It is associated with the current ActivityInstance, and needs the following input parameters: caseID (the currently executed process instance ID), activityID (the activity instance ID that has just terminated), and the conditionParameters (the values required by the conditions to be evaluated). The Next unit finds all the process constraints related to the specified activity instance, evaluates them according to the defined precedence constraints (i.e., sequence, AND-join, etc.), and, if the conditions hold, enables the execution of the subsequent activities in the workflow. If the activities are automatic, they are immediately started. If they involve human choice, the application model consists of the site view for the user to choose when to start the activity. An example of Next unit can be found in Section 8, dealing with Search-based Web applications.

For supporting the design of workflow-driven Web applications, several **tool extensions** have been prototyped and are currently being ported to the commercial version of WebRatio:

- A workflow modeling editor for specifying business processes in the BPMN notation.
- Model transformations that translate a business process model into a skeleton of WebML hypertext model.
- The abovementioned operation units and special-purpose links for implementing the static and dynamic workflow enactment.

5 User Personalization and Context Awareness

WebML has been also applied to the design of adaptive, context-aware Web applications, i.e. applications which exploit the context and adapt their behaviour to usage conditions and user's preferences [6].

In these applications, the design process is extended by a preliminary step dedicated to the **modeling of the user profiles** and of the **contextual information**.

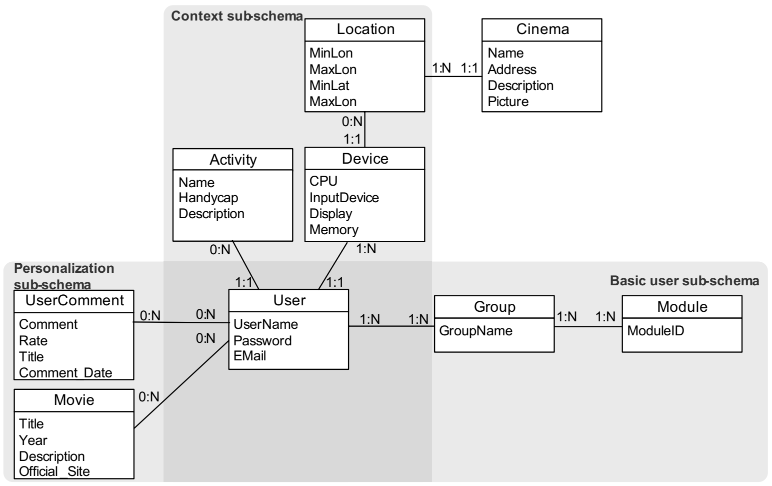


Fig. 7. Three models representing user, personalization, and context data

User and context requirements are described by means of three different models, complementing the application data (see Figure 7):

- The **user model** describes data about users and their access rights to the domain objects. In particular, entity *User* expresses a basic user profile, entity *Group* enables access rights for groups of users, and entity *Module* allows users and groups to be selectively granted access to any hypertext element (site views, pages, individual content units, and even links).
- The **personalization model** associates application entities with the *User* entity by means of relationships denoting user preferences or ownership. For example, the relationship between the entities *User* and *UserComment* in Figure 7 enables the identification of the comments s/he has posted, and the relationship between the entities *User* and *Movie* represents the preferences of the user for specific movies.
- The **context model** includes entities such as *Device*, *Location* and *Activity*, which describe context properties relevant to adaptivity. Context entities are connected to the *User* entity, to associate each user with her/his (personal) context.

During hypertext design, context-awareness can be associated with selected pages, and not necessarily with the whole application. **Location-aware pages** are tagged with a *C-label* (standing for "Context-aware") to distinguish them from conventional pages. Adaptivity actions are clustered within a *context cloud* which must be executed prior to the computation of the page. Clouds typically includes WebML operations that read the personalization or context data and then customize the page content or modify the navigation flow defined in the model.

A prototype extension of WebRatio generates pages with adaptive business logic; such a prototype has been used in some applications but has not been included yet into the commercial version.

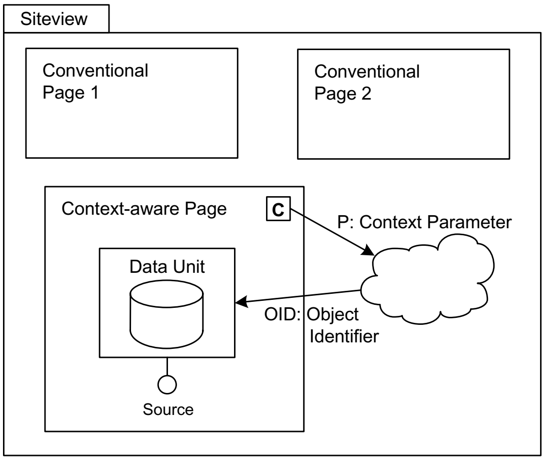


Fig. 8. Model with a context-aware page, labelled with a "C" and associated with a "context cloud"

6 Semantic Web Services

Traditionally, the service requestor and service provider are designed jointly and then tightly bound together when an application is created. The emerging field of Semantic Web Services (SWS) [26] provides paradigms for semantically enriching the existing syntactic descriptions of Web services; then, the service requestor can search, either at design or at run time, among a variety of Web-enabled service providers, by choosing the service that best fits the requestor's requirements. Such a flexible binding of requestor and providers allows for dynamic and evolving applications to be created, utilizing automatic resource discovery, selection, mediation and invocation.

We extended WebML in [4] so as to generate, on top of conventional models (of: processes, data, services, and interfaces), a large portion of the semantic descriptions required by the SWS in a semi-automatic manner, thus integrating the production and maintenance of semantic information into the application generation cycle.

To address the new SWS requirements, we defined a **process for semantic service design** by extending the SOA design process with two additional tasks:

- Ontology Importing, for reusing existing ontologies that may be exploited for describing the domain of the Web application under development.
- Semantic Annotation, for specifying how the hypertext pages or services can be annotated using existing ontological knowledge.

At the conceptual level, the **content model** for Semantic Web applications addresses the integration of existing third-party ontologies in the conceptual data model. At the logical level, imported ontological data can be either copied into an application-specific implementation of the E-R model (typically a relational database) or maintained in remote semantic repository and queried on demand.

The basic WebML primitives have been extended with components for **ontology querying and navigation**, exploiting the expressive power of ontological languages (inspired by SPARQL and RDF-S). These units allow queries on classes, instances, properties, and values; checking the existence of specific concepts; and verifying whether a relationship holds between two resources. Further units import content from an ontology and return the RDF description of a given portion of the ontological model. Operations such as lifting and lowering have been introduced too, by extending the XML2XML mapping components already developed in the context of SOAs. These units, together with the standard WebML primitives and the SOA extensions, allow designers to specify new kinds of applications. For instance, it is possible to define WSMO mediators [4], as demonstrated in the context of the SWS Challenge.

The SWS primitives have been implemented in two versions: when ontological data are maintained in an external repository, the implementation exploits ontological query languages; when ontological data are integrated within an internal relational source, the implementation is directly mapped to such source.

The WebRatio development tool has been extended with prototypical **automatic generators of WSMO-compliant descriptions** (goals, choreographies, capabilities, and mediators) from the models already available in WebML, i.e., business processes, content models, and service models. The automatically generated annotations cannot express the full semantics of services and applications, but they provide an initial skeleton, which can be completed manually.

7 Rich Internet Applications

Due to the increasingly complex requirements of applications, current Web technologies are starting to show usability and interactivity limits. Rich Internet Applications (RIAs) have been recently proposed as the response to such drawbacks; they are a variant of Web-based systems minimizing client-server data transfers and moving the interaction and presentation layers from the server to the client. While in traditional data-intensive Web applications content resides solely at the server-side, in the form of database tuples or as user session-related main memory objects, in RIAs content can also reside in the client, as main memory objects with the same visibility and duration of the client application, or even, in some technologies, as persistent client-side objects. Also, in RIAs more powerful communication patterns are possible, like server-to-client message push and asynchronous event processing. WebML has been extended with the aim of reducing the gap between Web development methodologies and the RIA paradigm, leveraging the common features of RIAs and traditional Web applications [3].

The design process is extended by defining the **allocation to the client or server side** of data elements (entities and relationships) and hypertext components (pages, content and operation units), and by establishing the relevant **client-server communication patterns** (consisting of policies for event notification, recipient filtering, and synchronous/asynchronous event processing).

In the **content model**, concepts are therefore characterized by two different dimensions: their **location**, which can be the server or the client, and their **duration**, which can be persistent or temporary. For example, in Figure 9 the *Wish Lists* entity is tagged as *client (C)* and temporary (unfilled icon) to denote that the data are temporarily stored at the client side, for the duration of the application run.

Similarly, the notion of page in WebML has been extended, by adding **client pages,** which incorporate content or logics managed (at least in part) by the client; their content can be computed at the server or client side, whereas presentation, rendering and event handling occur at the client side. The events generated by the user's interaction can be processed locally at the client or dispatched to the server. Event handling operations are also introduce (send event and receive event) which enable the expression of flexible communication patterns, including real-time

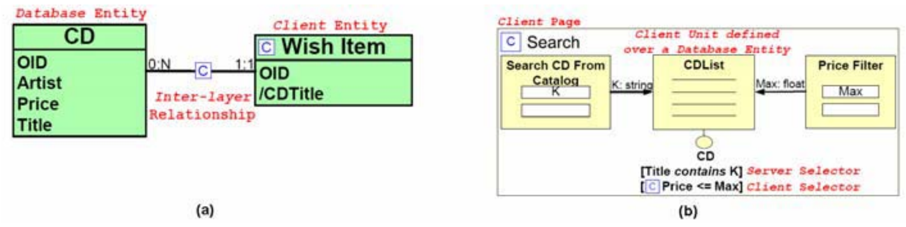


Fig. 9. Example of RIA-enabled WebML data (a) and hypertext model (b)

collaboration, server push, and asynchronous event processing. Classical WebML content units are also extended with the possibility of specifying that the source entity, the selector conditions, or ordering clauses be managed either on the server or on the client. Figure 9 shows a client page which contains an index unit with the population fetched from the server, but filtered using a predicate (price<=max) computed on the client. In order to fit the more flexible way in which RIAs handle the content of pages, the semantics of page computation has also been revised.

The RIA modelling primitives have been implemented in WebRatio through a prototypical **code generator for the client-side pages**, exploiting an open source RIA platform (www.openlaszlo.org) for handling events and managing the computation of client-side content units and operations. Each content unit is mapped into: (1) a view component for rendering, (2) a model component for data management, business logic, and server communication, (3) possibly a service on the server-side for data query and result formatting in XML. A subset of the prototyped features, including AJAX behaviours and client-side event management, is already available in the commercial version of WebRatio.

8 Related Work

Web Application Modeling. Several methodologies and notations address conceptual modeling of Web applications [31]. Among more recent projects, WebML is closer to those based on conceptual methodologies like W2000 [16] and OO-HMETHOD [29] (based on UML interaction diagrams), Araneus [35, 36], Strudel [27] and OOHDM [40]. The WAE UML extension by Conallen [23] focuses mainly on implementation and architectural issues of Web application design. Commercial vendors are proposing tools for Web development, however most of them have only adapted to the Web environment modeling concepts borrowed from other fields. Among them, Oracle JDeveloper 10g [38], Code Charge Studio [22], Rational Rapid Developer [39], and ArcStyler [13], which also features business process to Web model translation and direct implementation.

Business Processes. Several existing platforms and languages allow integrating the design of Web applications and business processes. Among the existing models, we can mention Araneus [36], that has been extended with a workflow conceptual model, allowing the interaction between the hypertext and an underlying workflow management system. The Process Modeling Language (PML) [37], a lightweight formalism similar to BPMN that can be automatically compiled into a simple Web-based application, starting from imperative programming-style syntax. Among the Web design proposals, OO-H and UWE have specifically addressed the integration of process and navigation modeling. The authors of OO-H and UWE propose a joint approach [34] to the integration of process and navigation modeling. In particular, both methodologies converge in the requirements analysis phase, where UML Use Case, Class, and Activity Diagrams are exploited to capture the requirements, and then, the methods slightly diverge in the design phase. In OOHDM [40], the content and navigation models are extended with activity entities and activity nodes respectively, represented by UML primitives. In WSDM [25], the process design is driven by the user requirements and is based on the ConcurTaskTrees notation.

Web Services. A flurry of activity is currently taking place in the field of Web service description [44]. Several XML languages for orchestration and choreography of services have been proposed (e.g., BPEL4WS [19]). WebML is expressive enough to capture any BPEL4WS-style service composition pattern [5]. The ActiveXML system [15] manages XML documents including calls to services, but ignoring Web interfaces and complex processes.

Semantic Web. Several traditional Web design methodologies (like OOHDM [41]) and new approaches (like Hera [43]) are focusing on Semantic Web applications. MIDAS is a framework based on MDA for Semantic Web applications [12]. Research efforts are converging on the proposal of combining Semantic Web Services (SWS) and Business Process Management (BPM) to create one consolidated technology, called Semantic Business Process Management (SBPM) [33]. Our extensions to the Semantic Web Services benefit from the WSMO [26] framework for handling Semantic Web Services.

Adaptivity and context-awareness. Within the domain of theWeb, so-called adaptive hypermedia systems [20] address advanced adaptation and personalization mechanisms, and recent research efforts also address the special needs of mobileWeb applications and portable device characteristics. HyCon [32], for example, is a platform for the development of context-aware hypermedia systems with special emphasis on location-based services. AHA! [24] is a user modeling and adaptation tool originally developed in the e-learning domain. Other works [18] address the fast development of context-aware (Web) applications along a technological, database-driven approach, combining a universal context engine in combination with a suitable content management system [30]. On top of the Hera project, Fiala et al. [2004] propose implementation and deployment of component-based, adaptive Web presentations. [17] extend the previous approach by addressing the lack of dynamism.

RIAs. Some approaches address the complexity of RIAs through the exploitation of state models for interface design. Exploiting MDA life-cycle is a missing feature in the related work. Our approach is also different with respect to other recent proposals in the Web Engineering field to represent the RIA foundations (e.g. [42]), because we include a more abstract level of representation of states and events.

9 Conclusions

The "WebML approach" has acted as a framework for continuous innovation and exploration of new research directions. This is made possible by a unique combination of environmental conditions:

- Availability of well-defined conceptual models;
- Extensibility of the model thanks to a plug-in based structure;
- Availability of a CASE tool for fast prototyping of application and easy integration of new features and components;
- Formally defined development process for Web applications;
- Strong link between the research (mostly performed in university) and the technology transfer into an industrial-strenght product (performed within a spin-off);

- Interactions with real world requirements, enabled by interaction with customers of the spin-off.
- Participation to the international research community, through experience and people exchange and several EU-funded projects.

This mix of ingredients has allowed us to follow our own pathway to innovation in conceptual modeling.

Wide adoption of the approach has been secured thanks to low learning barrier to the newcomers and availability of suitable tool support. Basic modelling skills are usually taught in 6 hours of academic lessons, and the full training program for certified professionals requires a total of 8 days. On-the-field statistics estimate that 2 to 3 months are needed for enabling full productivity at industrial level. Once this is achieved, high efficiency is granted in the development process, thanks to automatic code generation that reaches 90% of the total software artifacts in typical industrial applications [2].

Adoption of WebML can be inferred from the spreading of the associated WebRatio toolsuite. The WebRatio company registered more than 27,000 downloads of the tool since its first release (among them, more than 7,500 of the last Eclipse-based release, WebRatio 5). 70 companies adopted the commercial version of the tool for medium-to-large development projects and 158 universities subscribed to the academic program, that currently involves more than 5,900 students and researchers worldwide.

Acknowledgements

We wish to thank all the people who work for developing WebML within the "Database and Web" group at Politecnico di Milano and the developers of the Web Models spin-off.

References – WebML

[1] Acerbis, R., Bongio, A., Brambilla, M., Butti, S., Ceri, S., Fraternali, P.: Web Applications Design and Development with WebML and WebRatio 5.0. TOOLS, pp. 392–411 (2008), `http://www.webratio.com/`

[2] Acerbis, R., Bongio, A., Brambilla, M., Tisi, M., Ceri, S., Tosetti, E.: Developing eBusiness solutions with a model driven approach: The case of acer EMEA. In: Baresi, L., Fraternali, P., Houben, G.-J. (eds.) ICWE 2007. LNCS, vol. 4607, pp. 539–544. Springer, Heidelberg (2007)

[3] Bozzon, A., Comai, S., Fraternali, P., Toffetti Carughi, G.: Conceptual Modeling and Code Generation for Rich Internet Applications. In: International Conference on Web Engineering, pp. 353–360. Springer, Heidelberg (2006)

[4] Brambilla, M., Celino, I., Ceri, S., Cerizza, D., Della Valle, E.: Model-Driven Design and Development of Semantic Web Service Applications. ACM TOIT 8(1) (2008)

[5] Brambilla, M., Ceri, S., Fraternali, P., Manolescu, I.: Process Modeling in Web Applications. ACM TOSEM 15(4) (2006)

[6] Ceri, S., Daniel, F., Matera, M., Facca, F.: Model-driven Development of Context-Aware Web Applications. ACM TOIT 7(1) (2007)

[7] Ceri, S., Fraternali, P., Bongio, A.: Web Modeling Language (WebML): a modeling language for designing Web sites. WWW9 / Computer Networks 33 (2000)
[8] Ceri, S., Fraternali, P., Bongio, A., Brambilla, M., Comai, S., Matera, M.: Designing Data-Intensive Web Applications. Morgan Kaufmann, San Francisco (2002)
[9] Ceri, S., Matera, M., Rizzo, F., Demaldé, V.: Designing Data-Intensive Web Applications for Content Accessibility using Web Marts. Communications of ACM 50(4), 55–61 (2007)
[10] Comai, S., Fraternali, P.: A Semantic Model for Specifying Data-Intensive Web Applications Using WebML. In: Semantic Web Workshop, Stanford, USA (July 2001)
[11] Manolescu, I., Brambilla, M., Ceri, S., Comai, S., Fraternali, P.: Model-Driven Design and Deployment of Service-Enabled Web Applications. ACM TOIT 5(3) (2005)

References – Related Work

[12] Acuña, C.J., Marcos, E.: Modeling semantic web services: a case study. In: Proceedings of the 6th International Conference on Web Engineering (ICWE 2006), Palo Alto, California, USA, pp. 32–39 (2006)
[13] ArcStyler, http://www.arcstyler.com
[14] Brodie, M., Mylopoulos, J., Schmidt, J. (eds.): On Conceptual Modelling: Perspectives from Artificial Intelligence, Databases and Programming Languages. Springer, Heidelberg (1984)
[15] Abiteboul, S., Bonifati, A., Cobéna, G., Manolescu, I., Milo, T.: Dynamic XML Documents with Distribution and Replication, SIGMOD (2003)
[16] Baresi, L., Garzotto, F., Paolini, P.: From Web Sites to Web Applications: New Issues for Conceptual Modeling. In: Mayr, H.C., Liddle, S.W., Thalheim, B. (eds.) ER Workshops 2000. LNCS, vol. 1921, pp. 89–100. Springer, Heidelberg (2000)
[17] Barna, P., Houben, G.-J., Frasincar, F.: Specification of Adaptive Behavior Using a General-Purpose Design Methodology for Dynamic Web Applications. In: De Bra, P.M.E., Nejdl, W. (eds.) AH 2004. LNCS, vol. 3137, pp. 283–286. Springer, Heidelberg (2004)
[18] Belotti, R., Decurtins, C., Grossniklaus, M., Norrie, M.C., Palinginis, A.: Interplay of content and context. In: Koch, N., Fraternali, P., Wirsing, M. (eds.) ICWE 2004. LNCS, vol. 3140, pp. 187–200. Springer, Heidelberg (2004)
[19] BPEL4WS: Business Process Execution Language for Web Services, http://www.ibm.com/developerworks/Webservices
[20] Brusilovsky, P.: Methods and Techniques of Adaptive Hypermedia. User Model and User-Adapted Interaction 6(2-3), 87–129
[21] Castro, J., Kolp, M., Mylopoulos, J.: A requirements-driven development methodology. In: Dittrich, K.R., Geppert, A., Norrie, M.C. (eds.) CAiSE 2001. LNCS, vol. 2068, pp. 108–123. Springer, Heidelberg (2001)
[22] Code Charge Studio 2.3, http://www.codecharge.com/studio
[23] Conallen, J.: Building Web Applications with UML, October 2002. Addison-Wesley, Reading (2002)
[24] De Bra, P., Houben, G.-J., Wu, H.: AHAM: a Dexter-based Reference Model for Adaptive Hypermedia. In: HYPERTEXT 1999: Proceedings of the tenth ACM Conference on Hypertext and hypermedia: returning to our diverse roots, pp. 147–156 (1999)
[25] De Troyer, O., Casteleyn, S.: Modeling Complex Processes for Web Applications using WSDM. In: Third International Workshop on Web Oriented Software Technology, Oviedo 2003, pp. 1–12 (2003)

[26] Fensel, D., Lausen, H., Polleres, A., de Bruijn, J., Stollberg, M., Roman, D., Domingue, J.: Enabling Semantic Web Services – The Web Service Modeling Ontology. Springer, Heidelberg (2006)
[27] Fernandez, M.F., Florescu, D., Kang, J., Levy, A.Y., Suciu, D.: Catching the Boat with Strudel: Experiences with a Web-Site Management System. In: SIGMOD 1998, pp. 414–425 (1998)
[28] Fiala, Z., Hinz, M., Houben, G.-J., Frasincar, F.: Design and Implementation of Component-based Adaptive Web Presentations. In: ACM SAC, pp. 1698–1704
[29] Gómez, J., Cachero, C., Pastor, O.: Conceptual Modeling of Device-Independent Web Applications. IEEE MultiMedia 8(2), 26–39 (2001)
[30] Grossniklaus, M., Norrie, M.C.: Information Concepts for Content Management. In: WISE Workshops, pp. 150–159
[31] Fraternali, P.: Tools and Approaches for Developing Data-Intensive Web Applications: A Survey. ACM Computing Surveys 31(3), 227–263 (1999)
[32] Hansen, F.A., Bouvin, N.O., Christensen, B.G., Grønbæk, K., Pedersen, T.B., Gagach, J.: Integrating the Web and the World: Contextual Trails on the Move. In: Proc. of ACM-Hypertext 2004, pp. 98–107 (2004)
[33] Hepp, M., Leymann, F., Domingue, J., Wahler, A., Fensel, D.: Semantic Business Process Management: A Vision Towards Using Semantic Web Services for Business Process Management. In: Proceedings of the IEEE ICEBE 2005, Beijing, China, October 18-20, 2005, pp. 535–540 (2005)
[34] Koch, N., Kraus, A., Cachero, C., Melia, S.: Integration of Business Processes in Web Application Models. Journal of Web Eng. 3(1), 22–49 (2004)
[35] Mecca, G., Merialdo, P., Atzeni, P., Crescenzi, V.: The (Short) Araneus Guide to Web-Site Development. In: WebDB (Informal Proceedings), pp. 13–18 (1999)
[36] Merialdo, P., Atzeni, P., Mecca, G.: Design and development of data-intensive Websites: the Araneus approach. ACM TOIT 3(1), 49–92 (2003)
[37] Noll, J., Scacchi, W.: Specifying process-oriented hypertext for organizational computing. Journal of Network and Computer Applications 24, 39–61 (2001)
[38] Oracle, Oracle Developer Suite, JDeveloper 10g, `http://www.oracle.com/tools`
[39] Rational, Rational Rapid Developer, `http://www.ibm.com/software/awdtools/rapiddeveloper`
[40] Rossi, L., Schmid, H., Lyardet, F.: Engineering Business Processes in Web Applications: Modeling and Navigation Issues. In: Third International Workshop on Web Oriented Software Technology, Oviedo 2003, pp. 81–89 (2003)
[41] Schwabe, D., Rossi, G.: The Object-Oriented Hypermedia Design Model. Communications of the ACM 38(8), 45–46
[42] Urbieta, M., Rossi, G., Ginzburg, J., Schwabe, D.: Designing the Interface of Rich Internet Applications. In: Latin-American Conference on the WWW, pp. 144–153. IEEE, Los Alamitos (2007)
[43] Vdovjak, R., Frasincar, F., Houben, G.-J., Barna, P.: Engineering semantic web information systems in Hera. Journal of Web Engineering 2(1-2), 3–26 (2003)
[44] Web Services Description Language 1.1, W3C Note (March 2001)

GAMBUSE: A Gap Analysis Methodology for Engineering SOA-Based Applications

Dinh Khoa Nguyen[1], Willem-Jan van den Heuvel[1], Mike P. Papazoglou[1], Valeria de Castro[2], and Esperanza Marcos[2]

[1] European Research Institute in Service Science (ERISS), Tilburg University
P.O. Box 90153, 5000 LE, Tilburg, The Netherlands
{D.K.Nguyen,W.J.A.M.vdnHeuvel,M.P.Papazoglou}@uvt.nl
[2] Kybele Research Group, Rey Juan Carlos University
Tulipán S/N, 28933, Móstoles, Madrid, Spain
{valeria.decastro,esperanza.marcos}@urjc.es

Abstract. The objective of business service analysis is to identify candidate business processes and services, and provide an in-depth understanding of their functionality, scope, reuse, and granularity. Unfortunately, many of today's service analysis and design techniques rely on ad-hoc and experience-based identification of value-creating business services and implicitly assume a "blue sky" situation focusing on the development of completely new services while offering very limited support for discovering candidate services from a varied inventory of pre-existing software assets. In this article, we introduce a novel business service engineering methodology that identifies and conceptualizes business services in a business domain. Moreover, our approach takes into account a realistic situation, in which pre-existing enterprise assets must be considered for the reuse to implement fragments of the newly conceived business services. A running example is provided to exemplify our approach.[1]

1 Introduction

Service Oriented Architecture (SOA) promotes highly standardized, loosely coupled and Web-enabled services to foster rapid, low-cost and easy composition of distributed enterprise applications [17]. Central to any SOA development methodology are business service analysis and design techniques for identifying, conceptualizing, profiling and rationalizing service-enabled business processes. *Business services* capture transactional and value-creating business activities that are delivered by service providers to service clients under strict business conditions, e.g., sending an invoice, checking inventories or creating a purchase order [11]. Business services are intrinsically conceptual and implementation-agnostic. They may be physically realized in several alternative ways; typically however they are constructed with Web service technologies, e.g., SOAP, WSDL and BPEL. Unfortunately, many of today's service

[1] This work has been partially funded by the National Dutch Research Council (NWO) under the project Service-enAbling Pre-exIsting ENterpriSe Assets (SAPIENSA), and by the European FP-7 Network of Excellence S-Cube, Grant Agreement no. 215483.

analysis and design methodologies rely only on intuitive or experience-based service identification techniques that identify business-relevant services from existing best-in-class case studies. There is a lack of a more rigorous approach that ensures that the resulting business services can contribute high business value and have the right granularity level for maintainability and reusability purposes across the enterprise. The reuse of existing functionalities in the development of new business services in a specific business domain becomes one of the major aspects in modern SOA development methodologies. However, the migration of the existing systems to a new SOA environment goes far beyond the basic wrapping techniques, which are used to typically implement a thin SOAP/WSDL/UDDI layer on top of existing software functionalities [15]. While relatively simple non-critical applications may be effectively built in this way, designing industrial-strength SOAs necessitates a business service analysis methodology to support the dissecting, decomposing and repurposing existing software functionalities for the new business services.

The approach that we introduce in this article is named "Gap Analysis Methodology for BUsiness Service Engineering" (GAMBUSE) and attempts to address this specific shortcoming by concentrating on the following two inter-related issues:

- ***Business Service Identification***: In a service-enabled to-be process, each activity contributes to a well-defined business objective, e.g., order management, inventory management, customer management, etc., and could be realized by a high-level composite service interaction. GAMBUSE introduces a process-driven technique to identify and conceptualize meaningful business services for an SOA project by collectively grouping all activities that contribute to the same course into logically cohesive and loosely coupled business services. A high-quality SOA design must ensure that the resulting business services have the right level of granularity for the maintainability and reusability purposes.
- ***Gap Analysis approach for reusing and repurposing existing software assets***: GAMBUSE adopts a flexible gap analysis approach that detects and assesses the reuse of available software assets (termed as-is services) as (parts of) the newly conceived business services that collectively shape a to-be business process. Functional equivalence and behavioural and policy misfits between newly developed to-be process and (mined) as-is processes from existing physical enterprise assets are taken into account to support the reuse strategy of existing software functionalities.

Our contribution is the definition of a model-driven approach with *models* as the first-class citizens. Adopting a model-driven approach adds the benefits of the "Model Driven Architecture" (MDA) approach as defined by the OMG [13] to the development process of SOA. The MDA approach provides guidelines to capture the structural as well as the behavioural aspects of a SOA using models, which can then be translated via standardised transformation rules into interface specifications for any particular application platforms. Over the last years the OMG has recognized that not all applications can be developed in a top-down manner and introduced the "Architecture Driven Modernisation" (ADM) approach [24] that provides methods to recover design information from existing applications for which no MDA model exists (bottom-up approach).

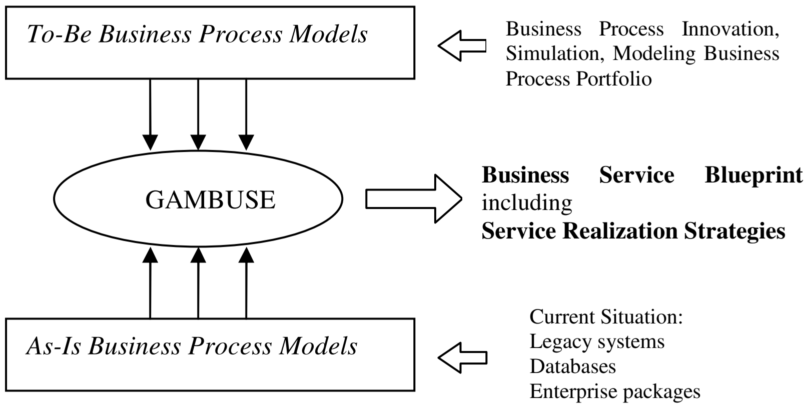

To-Be Business Process Models
Business Process Innovation, Simulation, Modeling Business Process Portfolio
GAMBUSE
Business Service Blueprint including Service Realization Strategies
As-Is Business Process Models
Current Situation:
Legacy systems
Databases
Enterprise packages

Fig. 1. Inputs and Output of GAMBUSE

The GAMBUSE approach is shown in Fig. 1. Inputs of GAMBUSE include the abstract *as-is and to-be process models.* The as-is process models are recovered from pre-existing software assets, e.g., legacy systems, databases, enterprise packages, etc., and relies on the ADM approach[2] allowing the various stakeholders to understand the portfolio of available applications and business processes. In contrast, the to-be process models are the desired novel business processes introduce to address current and future enterprise needs. Output of GAMBUSE is a *business service blueprint* that can be used to define abstract business services for the next service design phase. This blueprint not only specifies the new business services in terms of their functionality, behaviour and policies, but also defines several service realization strategies and proposes a decision framework for selecting an appropriate one. A *service realization strategy* should include design decisions whether (parts of) the new business service functionalities can be obtained by reusing or revamping existing (internal or external) IT assets, or whether they should be re-designed or developed from scratch [15,17].

In general, gap analysis has already been extensively studied in various domains such as business process management [9,14] and supply chain management [5]. Several approaches have emerged which aim to help organizations obtain a better understanding of improving their business. However, it is important to remark that although most of these proposals provide accurate techniques and indicators to better understand the best practices leveraging business process improvements, they are unfortunately not tailored towards specific requirements of SOA processes and leave service-development in the users' hands. To the best of our knowledge, there exists no proposal that defines a model-driven approach for gap analysis, although it is possible to identify some work describing, in a very abstract way, some details that need to be taken into account in order to drive the gap analysis by means of models [2,19].

The remainder of this article is structured as follows: Section 2 explains our gap analysis methodology GAMBUSE in detail. A running example is provided in

[2] This depends on the level of documentation available in an enterprise. If the process model descriptions are detailed and well-documented, an ADM approach is not necessary.

Section 3 to exemplify our approach. Section 4 introduces our method to formally describe a service (or process) based on a stratified reference service meta-model, on which the whole GAMBUSE methodology relies. In Section 5, we scrutinize step 3 of the GAMBUSE methodology, namely the technique to identify and extract business services out of the to-be process models. Section 6 targets the matching technique in GAMBUSE that amalgamates various existing techniques. Lastly, Section 7 concludes the article by presenting open research issues to be tackled in future work.

2 GAMBUSE - A Model-Driven Gap Analysis Methodology for Business Service Engineering

The GAMBUSE approach for gap analysis is firmly grounded on model management, which has been proposed for studying a similar problem, albeit in the domain of data integration [3,4]. In contrast to the traditional data-oriented approach that focuses on data heterogeneity- and impedance mismatches, GAMBUSE leverages a number of *generic meta-model operators*[3] to address its two ultimate goals of the business service analysis: ***Business Service Identification*** and ***Gap Analysis***.

To achieve canonicalization and consequently alleviate problems associated with terminology nuances and functional granularity, we assume that both the as-is and to-be models are grounded on the Supply Chain Operations Reference model SCOR [5]. In particular, we assume that the GAMBUSE as-is and to-be models are expressed in SCOR Level-4 processes. Level-4 processes allow enterprises to implement their own specific supply-chain management practices. Level-4 processes extend Level-3 processes that provide standard process element definitions, process element information, inputs and outputs, process performance metrics and best practices where applicable in a specific supply chain. Furthermore we assume that the process and activity names are consistent in both the as-is and to-be models. The specific steps of service gap analysis are depicted in Fig. 2.

- **Step 1: Creating meta-model instances for the as-is and to-be business process.** GAMBUSE is based on a stratified reference Service Meta-Model (SMM) that specifies and correlates all modeling constructs for business processes. During this step, the Service Schema Specifications (SSS) of the as-is and to-be process that contain their activities, business entities, attributes, constraints, business rules, etc., are instantiated from the SMM. Inspired by the model management vision in [3,4], the SMM accommodates the *generic meta-model operators* essential for deriving functional fragments (in step 2) and subsequently business services (in step 3) from the to-be process model, as well as for the matching between as-is and to-be process in step 4. Section 4 will first describe the SMM in detail, and then explain how to instantiate the SSS from the SMM for the as-is and to-be process.

[3] A meta-model operator is a logical operator that works on the instances of a meta-model. It can be a simple set operator, or a complex high-level one. Sample operators are `extract`, `insert`, `Unify`, `Decompose`, `Subtract` or `Scope`. See Appendix A for a comprehensive overview of operators in GAMBUSE.

- **Step 2: Identification and scoping of to-be functional fragments**. In the second step conceptual boundaries are conceived around modelling elements from the to-be model, mining modelling fragments that are rudimentary formations of related conceptual model elements. Therefore, GAMBUSE adopts the technique from [23] in parsing a business process into a Process Structure Tree containing Single-Entry-Single-Exit (SESE) canonical[4] fragments and residual activities. The meta-model operators work on the instances of the to-be activities and produce the new functional SESE fragments that are again instantiated from the SMM for the next step.
- **Step 3: Distilling business services from the to-be process model.** The identified and scoped SESE functional fragments are used as inputs for the identification of business services. This step subsequently applies *meta-model operators* on the functional fragments to yield business services that comply with essential quality criteria including high functional and logical cohesion, low coupling, and the right level of scoping and autonomy (granularity). Section 5 will describe and illustrate this step in detail with a running example.
- **Step 4: Detecting and assessing the reusability of as-is systems.** Functional overlaps and discrepancies, and misfits in behaviour and business policies between the as-is and to-be process are discovered during step 4. This step aims to detect reusable as-is functionalities (exposed by the existing software systems to implement as-is activities) for implementing (parts of) the newly identified business services in the to-be process. A matching method is used in this step to yield the logical equivalence between the as-is and to-be activities. Section 6 will explain the matching in detail by means of a running example.
- **Step 5: Service Realization and Reusability strategy.** During this step, the list of business services is consolidated, and subsequently an appropriate realization strategy is selected. Strategies include the reuse of existing assets through repurposing or revamping, development from scratch, outsourced development contract, service

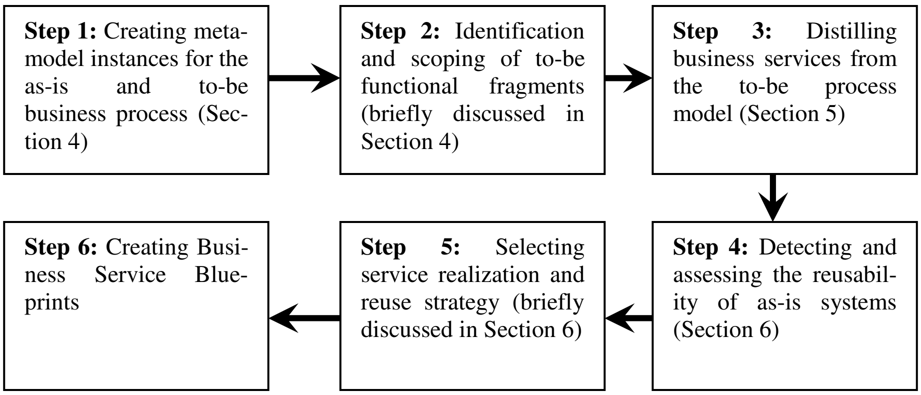

Step 1: Creating meta-model instances for the as-is and to-be business process (Section 4)
Step 2: Identification and scoping of to-be functional fragments (briefly discussed in Section 4)
Step 3: Distilling business services from the to-be process model (Section 5)
Step 4: Detecting and assessing the reusability of as-is systems (Section 6)
Step 5: Selecting service realization and reuse strategy (briefly discussed in Section 6)
Step 6: Creating Business Service Blueprints

Fig. 2. GAMBUSE steps

[4] Canonical means that the SESE fragment must be maximal. More explanations can be found in [23].

rental contract, infrastructure rental contract, etc [15,17]. A migration plan is devised that will ease the transformation of the existing assets in the as-is model into the new business services.

- **Step 6: Creating Business Service Blueprints.** GAMBUSE concludes with the construction of an abstract business service model and detailed metadata that captures conceptual processes and policy definitions, and serves as the basis for further concretization into service interface definitions (e.g., using WSDL, WS-Policy and BPEL) during service design. This phase is by now well understood and its detailed description can be found in [17]. Note that the transformation from the abstract to the physical model is still under research, but may already be facilitated by several model-driven techniques, including [10].

In the remainder of the article, we will explain the steps of GAMBUSE in Fig. 2 more in detail, by means of a case study that will be outlined in the next section.

3 Case Study

We introduce in this section a simple, yet realistic case study to exemplify our gap analysis approach. Let AutoScore be a fictional automotive company that builds its supply chain conforming to the SCOR model.

Fig. 3 illustrates the as-is and to-be SCOR level 4 models implementing the SCOR level 3 activity **D1.2** *"Receive, Validate and Enter Order"* by the company AutoScore. The as-is process (on the left side of Fig. 3) is the abstraction of an activity flow that is subsequently executed by the existing functionalities exposed by existing software systems. Normally, as-is enterprise systems communicate with each other via standardized synchronous invocation such as Remote Procedure Call (RPC) or Remote Method Invocation (RMI) in Java, Remote Function Call (RFC) with Business API (BAPI) within SAP systems, or by using Sapjco (Sap-java-connector) library for a connection between a Java system and a SAP system. In some special cases, especially in the shop-floor manufacturing systems like the Programmable Logic Controller or Manufacturing Execution System, communication is only supported by proprietary APIs.

From the technical design point of view, a problem from which enterprises must suffer is the necessity to support various communication protocols among heterogeneous systems. Other inefficiencies are the maintainability and reusability of IT assets. Software functionalities are hard to maintain and reuse across the enterprise since they statically belong together to enterprise packages or legacy systems and may also strongly depend on each other. Furthermore, flexibility and agility are other significant issues for market competitiveness that enterprises cannot fulfil with this static setting. For instance, changes in the process because of new market demands lead to necessary modifications of legacy software systems that could take months to complete and by that time enterprises may have already lost the market.

From the business analysis point of view, the as-is process in Fig. 3 reflects a basic process variant that suffers from several inefficiencies:

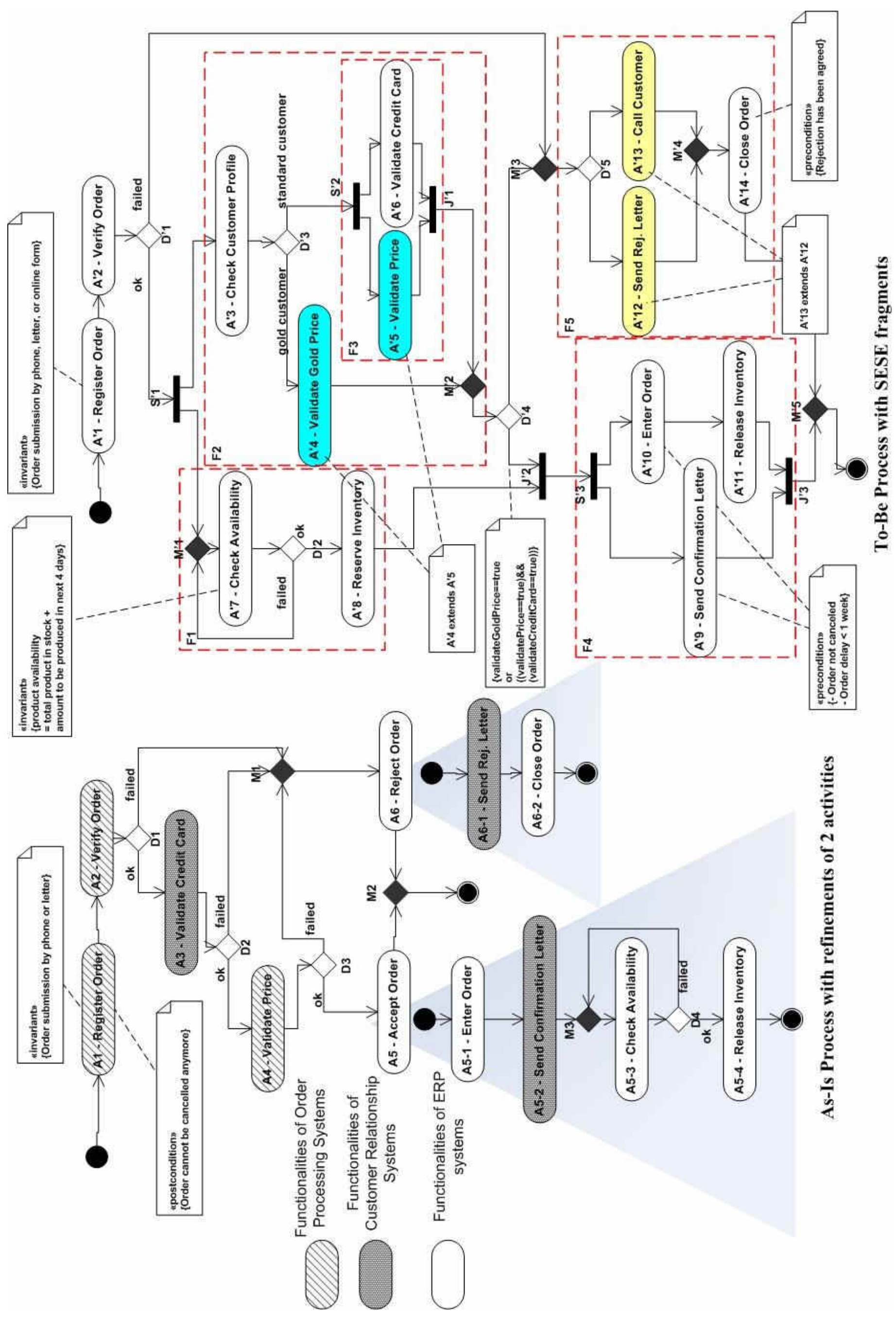
- Order submission is possible only via phone or letter, because the Order Processing system does not provide a web interface.
- Order cancellation is not allowed after submission. Order Processing and ERP systems do not have the 'revert' option.
- The process does not distinguish between different kinds of customers, treating all orders in the same manner.
- Most activities are executed sequentially, although they are not dependent on each other, e.g., *A3* and *A4*.
- Product availability is checked very late (*A5-3*), even after the order confirmation is sent to the customer (*A5-2*). In the worst case that the product volume at that time is not enough for order fulfilment and still needs extra time to be produced, the order might be delayed longer.
- Rejecting customers' orders with letters might not be convenient for explaining the reasons and solving disputes.

The as-is process in Fig. 3 is recovered by the IT specialists from existing running systems, reflecting process details on several granularity levels. For instance, in our case study activities *A5* and *A6* are described with more detailed refinements containing the nested finer-grained activities. These nested activities can be considered to be in AutoScore's SCOR level 5 process configuration or below.

In addition, we assume AutoScore has implemented a Java system for Order Processing and utilizes a legacy CRM system and an ERP package. Fig. 3 also explains which activities are executed by which of these 3 systems. An ad-hoc process redesign and reconfiguration among these systems may help target the inefficiencies mentioned above, but it would lead to enormous effort and cost to change the current static setting. Moreover, AutoScore foresees the necessity to reconfigure their systems into a more modular, dynamic and loosely coupled paradigm with standardized communication protocols to cope with future unforeseen demands.

This situation has lead to the decision of AutoScore to use an SOA to rationalize their system landscape. Alongside, business analysts of AutoScore sit down together and improve (redesign) this order process based on some industrial best practices, e.g., SCOR best practices. The redesigned to-be process will be implemented following the SOA paradigm. Firstly, the technology-independent process model on the right side of Fig. 3 is designed by the business analysts using UML activity diagram. The next step is our focus to show how GAMBUSE works to extract meaningful business services from the to-be model and to assess the reuse of existing as-is functionalities for implementing (parts of) these new business services. In the last step, all the process and business service models will be translated into technology-specific models, e.g., into WS-* and BPEL, so that they can be deployed and turn into operation mode.

A simplifying assumption of our running example is that regardless of their level of granularity, activities within AutoScore always have unique and meaningful labels. This holds for both as-is and to-be processes. The to-be process allows customers to submit an order via online form, and to cancel an order if it has not been confirmed (*A'1*). Furthermore it differentiates between gold and standard customers. The customer type of an individual client can be checked with the customer profile (*A'3*). Gold customers' credits are fully trusted and hence do not need to be validated.



To-Be Process with SESE fragments
A'1 - Register Order
A'2 - Verify Order
A'3 - Check Customer Profile
A'4 - Validate Gold Price
A'5 - Validate Price
A'6 - Validate Credit Card
A'7 - Check Availability
A'8 - Reserve Inventory
A'9 - Send Confirmation Letter
A'10 - Enter Order
A'11 - Release Inventory
A'12 - Send Rej. Letter
A'13 - Call Customer
A'14 - Close Order
«invariant» {Order submission by phone, letter, or online form}
«invariant» {product availability = total product in stock + amount to be produced in next 4 days}
A'4 extends A'5
A'13 extends A'12
{validateGoldPrice==true or ((validatePrice==true)&& (validateCreditCard==true))}
«precondition» {- Order not canceled - Order delay < 1 week}
«precondition» {Rejection has been agreed}
standard customer
gold customer
ok
failed
F1
F2
F3
F4
F5
As-Is Process with refinements of 2 activities
A1 - Register Order
A2 - Verify Order
A3 - Validate Credit Card
A4 - Validate Price
A5 - Accept Order
A6 - Reject Order
A6-1 - Send Rej. Letter
A6-2 - Close Order
A5-1 - Enter Order
A5-2 - Send Confirmation Letter
A5-3 - Check Availability
A5-4 - Release Inventory
«invariant» {Order submission by phone or letter}
«postcondition» {Order cannot be cancelled anymore}
Functionalities of Order Processing Systems
Functionalities of Customer Relationship Systems
Functionalities of ERP systems

Fig. 3. A Case Study in SCOR level 4

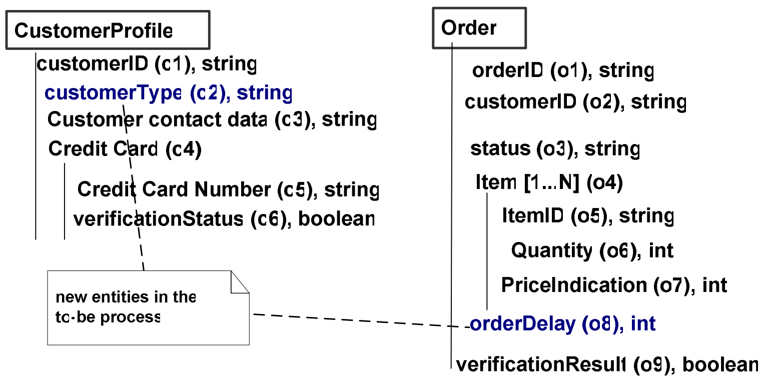

CustomerProfile
customerID (c1), string
customerType (c2), string
Customer contact data (c3), string
Credit Card (c4)
Credit Card Number (c5), string
verificationStatus (c6), boolean
new entities in the to-be process
Order
orderID (o1), string
customerID (o2), string
status (o3), string
Item [1...N] (o4)
ItemID (o5), string
Quantity (o6), int
PriceIndication (o7), int
orderDelay (o8), int
verificationResult (o9), boolean

Fig. 4. Data Structures of Business Entities and Information Entities used in the case study

Assuming customers have been informed about the item prices in the product catalogue and hence, they must indicate the item prices in the order and these prices are validated against an exclusive tariff scheme for gold customers (*A'4*). In contrast, credit cards of standard customers are checked for validity before the order can be processed (*A'6*), and the prices are validated against a standard tariff scheme (*A'5*). Both activities are independent and may be executed in parallel. Note that the activity *A'4* is marked by the business analysts as an extension of *A'5* and can accept different types of arguments, i.e., both gold and standard tariff schemes.

In addition, business analysts have concluded that availability checking can be improved by rescheduling this activity (*A'7*), and introducing an auxiliary inventory reservation activity (*A'8*). After inventory has been checked and reserved, the order can be accepted, and a confirmation letter is issued to the customer (*A'9*), while entering the order into the system (*A'10*), and releasing product inventory for the delivery (*A'11*). In the revamped process, not only a rejection letter (*A'12*) can be issued, but also the customers can be called to be informed of the rejection (*A'13*). Again, business analysts have marked *A'13* as an extension of *A'12*. In the last step of the process, the order is finalized and administratively logged (*A'14*).

Within the as-is and to-be processes, control nodes are modelled explicitly and assigned also with unique labels. Xor-type decision nodes have the name beginning with D, Merge nodes with M, Split node with S and Join node with J. Pre-, post-conditions and invariants for each activity are also specified by the annotations or expressed by the guards of activity transitions. Global business rules and regulatory compliance can be expressed by following a particular flow of activities, or can be translated into local invariants that constrain on every activity.

Beside the process model, business analysts also specify the data structures of the Information Entities used within the process. Fig.4 depicts the Information Entities that capture input and output data of the process activities. They can have not only primitive types like integer, boolean, string, etc., but also complex types, meaning that they can contain other nested complex and simple types. Information Entities are gathered together to higher-level Business Entities. For instance in Fig. 4, the Business Entity *Order* comprises of *orderID (o1), customerID (o2), status (o3)*, a list of

Items (o4), a *orderDelay(o8) and a verificationResult(o9).* An *Item (o4)* is an Information Entity of a complex type containing *ItemID (o5), Quantity(o6)* and *PriceIndication(o7).* Note that in the to-be process, business analysts decide to reuse the existing as-is data structures and may introduce new Information Entities or redefine the existing ones. For instance in the to-be process, *customerType* is newly introduced to distinguish different kinds of customers, and *orderDelay* is introduced to keep track of the pending time of the order.

Step 1 of GAMBUSE instantiates the as-is and to-be processes including their activities, business entities, information entities, constraints, rules etc. from the SMM. The results of step 1 will be two SSS descriptions for the as-is and to-be process respectively. Details of the SMM, SSS and instantiating examples of our case study will be introduced in the next section.

4 Core Service Meta-Model (SMM) and Service Schema Specification (SSS) for GAMBUSE

GAMBUSE is developed on top of a stratified reference *Service Meta-Model* (SMM) that specifies and correlates modelling constructs for processes, policies and business services. For each specific as-is and to-be business process, a Service Schema Specification (SSS) can be generated from the SMM by instantiating the concepts and relations in the SMM. The SSS will be further used to generate executable service instances that are deployed and operate in some process and service engines. Fig. 5 illustrates the relationships between these abstraction levels. However, in this article we neglect the instance level of processes and services since GAMBUSE only concentrates on the processes and services at the design time. This section will introduce the details of the SMM (Section 4.2), and the formal description of the concepts in SMM and elements in SSS (Section 4.3), on which our meta-model operators can be applied.

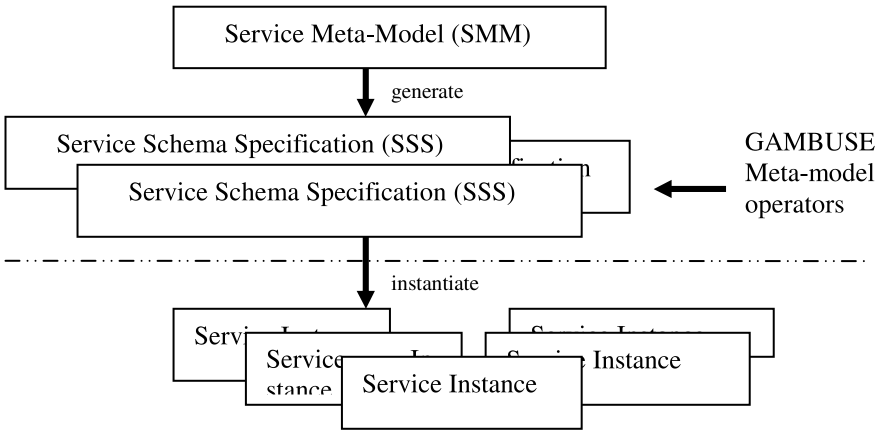
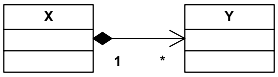

Service Meta-Model (SMM)
generate
Service Schema Specification (SSS)
Service Schema Specification (SSS)
GAMBUSE Meta-model operators
instantiate
Service Instance
Service Instance

Fig. 5. Abstraction levels in GAMBUSE

4.1 Universal Relationships

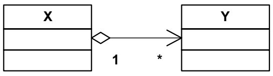
In order to show relationships between concepts in the SMM we have adopted the formal semantics in [1] for describing relationships such as composition, aggregation, and association found in object-oriented languages and in the AI semantic nets:

Composition: $\forall y, \exists! x: \overset{c}{\overrightarrow{xy}}$



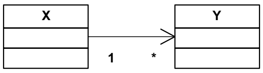
The composition relation between x and y indicates that if x is deleted, y will be deleted too. Between two concepts x and y there can be only one composition relation.

Aggregation: $\forall y, \exists x: \overset{a}{\overrightarrow{xy}}$



Between x and y there could be several aggregation relations. An aggregation relation between x and y indicates that if x is deleted, y will be deleted too, if y has no other relations.

Association: $\exists y, \exists x: \overset{s}{\overrightarrow{xy}}$



An association relation does not pose any restrictions on x and y.

In general we define a relation between x and y with the following description

General relation: $relation = (name, \overset{r}{\overrightarrow{xy}}) \quad r = \{c, a, s\}$

In UML, relations between the concepts have also the cardinality. We define the cardinality with the following notation:

$| cardinality_x || cardinality_y |^{-1}, \qquad cardinality \in \{0 | \geq 0 | 1 | \leq 1 | \geq 1\}$

where
0 means the cardinality = 0
>=0 means cardinality = 0…*
1 means cardinality = 1
<= 1 means cardinality = 0…1
>= 1 means cardinality = 1…*

For example if we have the following description of a relation R between 2 concepts x and y, $(R, \overset{a}{\overrightarrow{xy}} | 1 || \geq 1 |^{-1})$, it means that x is an aggregation of one or many (1…*) y.

4.2 Service Meta-Model (SMM)

The SMM adapted from [1] comprises of concepts, and their parameters and relations. In particular, parameters can be property domains or attributes. A property domain pd_i

has a set of properties p_i. The generation of the SSS from the SMM will assign a specific property for a property domain. For example, the message pattern (property-domain) of an (WSDL) service operation will be set to request-response (property). Attributes are the parameters which are assigned specific values when a service instance is created from the SSS, i.e., during runtime. In this article, we do not discuss the instance level. Hence, although attributes are provided in our formal descriptions, they are not considered in our approach.

Fig. 6 illustrates the SMM using classical UML class diagram notations. Each class (representing a concept) in Fig. 6 can be described with the following formal notation

$$Concept(\ property - domain^*, attribute^*, relation^*)$$

For instance, the concept *Information Entity* in Fig. 6 has the following description[5]

$$Information\ Entity(type, valueRange, Att^*, (includes, Message\ \overset{\xrightarrow{s}}{InformationEntity}[\geq 0][\geq 0]^{-1}),...)$$

which means that the concept *Information Entity* has two property domains *type* and *valueRange*, and among other relations, an association *"includes"* with the concept *Message*. The relation indicates that a *Message* might include 0 or many *Information Entities* and an *Information Entity* can be included in 0 or many *Messages*. The concept *Information Entity* might have a set of attributes *Att** but we do not list them here. In the following, other important concepts and relations in SMM will be described.

The SMM spans three logical layers: `Structural`, `Behavioural`, and `Regulatory`. A `Service` constitutes the heart-and-soul of the SMM, crosscutting the three layers. They can be a high-level business services, a composite application services, an atomic technical service, or a human-provided service.

The `structural` layer defines static description of business service constituents and its operational semantics. In particular, this layer specifies `service operations`, i.e., specific functions that a `Service` provides by consuming input `messages` and producing output `messages`. `Messages` are logically organized in typed `Information Entities`, which collectively make up `Business Entities`.

Next, the `behavioural` layer is designed on top of the `structural` layer, managing control and execution aspects of the `Services`. In particular, this layer predefines `(Sub-)Processes` that are the implementations of business services. They comprises of `functional fragments`, each of which captures a cohesive capsule of business functionality, such as invoicing. `Functional fragments` encompass a number of process `activities`. An `Activity` entails a logical functionality that invokes an (atomic or composite) `Service Operation` of another `Service`, or it can be a control flow construct like decision, merge, fork and join, or it can simply be a human task. (Sub-)`Processes` are designed as state-full, where the `activities` and `functional fragments` are `state transitions` that may query or change attribute values of `Business Entities`. This

[5] Note that we have omitted here some other relations of the concept *Information Entity* for brevity.

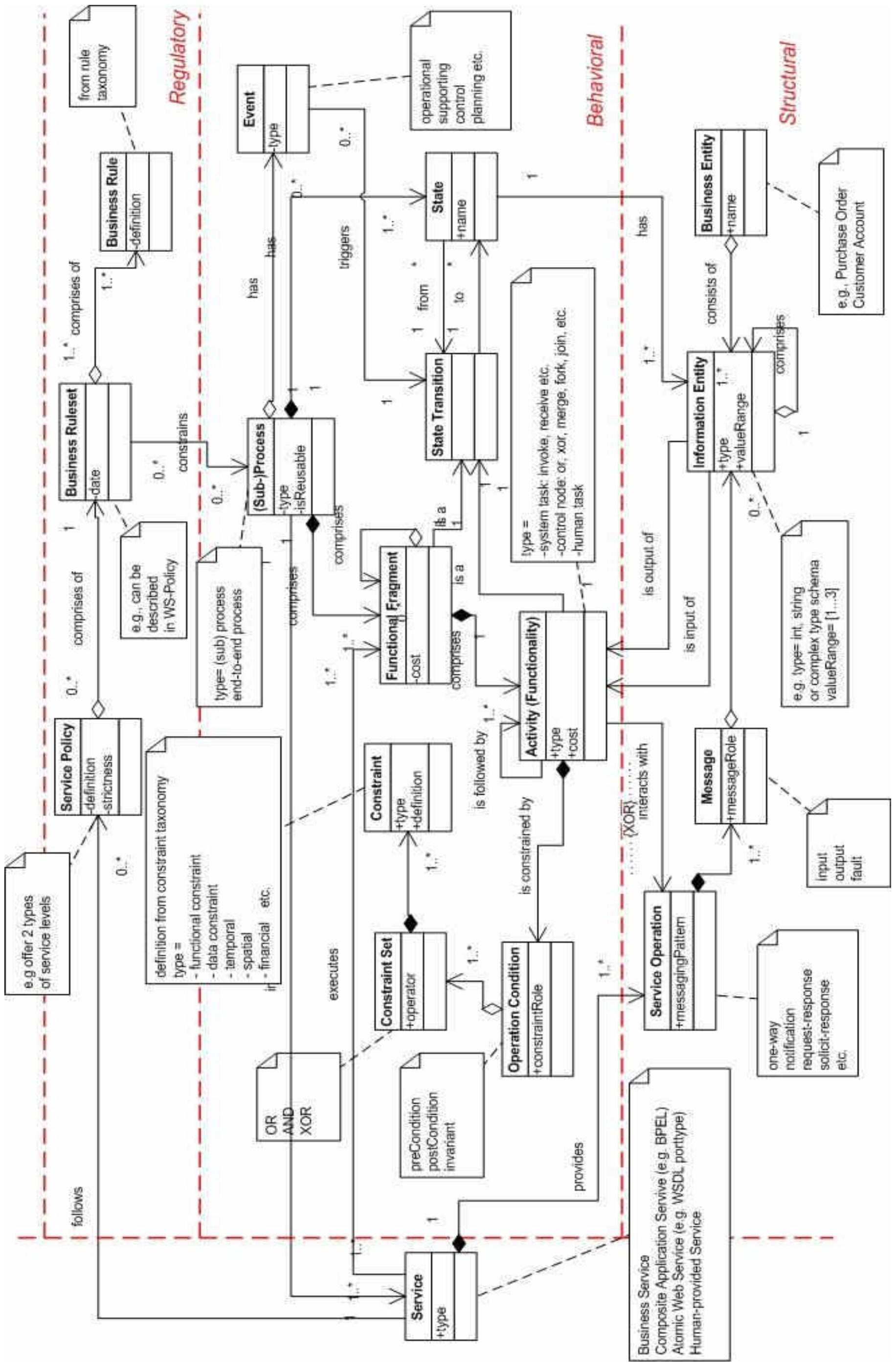

Regulatory
Behavioral
Structural
Business Rule
-definition
from rule taxonomy
Business Ruleset
-date
e.g., can be described in WS-Policy
Service Policy
-definition
-strictness
e.g offer 2 types of service levels
Event
-type
operational supporting control planning etc.
State
+name
State Transition
(Sub-)Process
-type
-isReusable
type= (sub) process end-to-end process
Functional Fragment
-cost
Activity (Functionality)
+type
+cost
type = -system task: invoke, receive etc. -control node: or, xor, merge, fork, join, etc. -human task
Constraint
+type
+definition
definition from constraint taxonomy type = - functional constraint - data constraint - temporal - spatial - financial etc.
Constraint Set
+operator
OR AND XOR
Operation Condition
+constraintRole
preCondition postCondition invariant
Business Entity
+name
e.g., Purchase Order Customer Account
Information Entity
+type
+valueRange
e.g. type= int, string or complex type schema valueRange= [1...3]
Message
+messageRole
input output fault
Service Operation
+messagingPattern
one-way notification request-response solicit-response etc.
Service
+type
Business Service Composite Application Servive (e.g. BPEL) Atomic Web Service (e.g. WSDL porttype) Human-provided Service
comprises of
constrains
has
triggers
from
to
comprises
is a
is followed by
is constrained by
executes
follows
provides
interacts with
{XOR}
is output of
is input of
consists of

Fig. 6. Service Meta-Model (SMM)

layer also defines `Operation Conditions` that can be clustered into `constraint sets` and then attached to `activities`, as pre- and post-conditions and invariant including, temporal, spatial, financial, constraints.

Lastly, the `regulatory` layer conceptualizes business decision `rules` as well as `policies` that govern the business logic of process. Particularly, this layer defines the concept of a `Business Ruleset`, capturing user-defined `business rules`. These rules constitute an abstract `Service Policy` that all `Services` must respect.

4.3 Service Schema Specification (SSS)

A Service Schema Specification is a formal description generated from the SMM for a specific service or process. Business processes themselves can be considered as high-level services and hence can be transformed into SSS descriptions. In step 1, our GAMBUSE approach leverages the SSS as a means for formally describing the as-is and to-be processes in a way that meta-model operators can work on them later.

Definition 4-1. ***Service Schema Specification (SSS)*** [1]: SSS is a collective set of elements $\mathcal{E} = \{e_i\}, \quad i = 1,...,n$, in which each e_i is an instance of a *concept in the SMM* and described by a tuple in the following BNF-format:

$$
\begin{aligned}
& e := < name, attribute^{*}, (property^{?}, relation^{?})^{*}, type > \\
& name := \text{string}, \quad attribute := \text{string}, \quad property := \text{string}, \\
& name := val(Concept), \quad property := val(property - domain) \\
& type := \text{as - is} \mid \text{to - be}, \\
& relation := (name, RT_e \mid c \parallel c \mid^{-1}) \\
& RT_e := \overset{c}{\overrightarrow{ee}} \mid \overset{a}{\overrightarrow{ee}} \mid \overset{s}{\overrightarrow{ee}} \\
& c := \text{integer} \\
& \text{'*' means 0 - n occurrence s, and '?' means 0 - 1 occurence.}
\end{aligned}
$$

An element *e* is an instance of a concept in the SMM, and according to our assumption, it must have a globally unique and expressive name. The instantiation assigns a unique *name* for the element and selects a specific *property* for each *property-domain* by applying the function *val(). Type* indicates whether this element belongs to the existing as-is process or the redesigned to-be process. *Properties* can also constrain the instantiated *Relations*, e.g., if a *Service Operation* instance has the *messagePattern = request-response*, then it must have only one *Message* with *messageRole = input* and one *Message* with *messageRole=output.*

Step 1 of GAMBUSE creates two SSSs for the as-is and to-be process model respectively by instantiating all corresponding model elements, such as the processes, activities, input/output Information Entities, Operation Conditions, Constraints, etc. We briefly introduce in the following the formal description of the element *A'4* in the to-be SSS, which represents the to-be activity *A'4 (Validate Gold Price)* in our case study in Section 3 (Fig. 3).

A'4 is an instance of the SMM concept `Activity` that has the following formal description:

$$\mathit{Activity}(\mathit{type}, \cos t, \mathit{Att}^*, ..., (\text{'is followed by'}, \mathit{Activity} \xrightarrow{s} \mathit{Activity}[\geq 0][\geq 0]^{-1}),$$

$$..., (\text{'is input of'}, \ \mathit{InformationEntity} \xrightarrow{s} \mathit{Activity}[\geq 0][\geq 0]^{-1}), ...)$$

Note that we have omitted some other relations of the SMM concept `Activity` for the reason of brevity. Then, *A'4* is instantiated from the concept `Activity` by assigning specific name, properties, and details of the relations, according to our case study in Section 3 (Fig. 3 and Fig. 4). The instantiated cardinalities of the relations in *A'4* must be the same or restrictions of the cardinalities of the relations in `Activity.` Afterwards, *A'4* becomes an element in the to-be SSS and has the following formal description:

$$A'4 :=< \text{'Validate Gold Price'}, \mathit{Att}^*,$$

$$(\text{'system task'}, (\text{'is followed by'}, A'4 \xrightarrow{s} M'2[1][1]^{-1}), (\text{'is followed by'}, D'3 \xrightarrow{s} A'4[1][1]^{-1})), 50,$$

$$......, (\text{'is input of'}, \ \mathit{IEo}7 \xrightarrow{s} A'4[\geq 1][1]^{-1}), ... >$$

where M'2, D'3 are two other instances of the SMM concept Activity and IEo7 is an instance of the SMM concept Information Entity

$M'2 :=< \text{'Merge - Node To - be 2'}, \mathit{Att}^*, (\text{'control node'}, ...)...... >$

$D'3 :=< \text{'Decision - Node To - be 3'}, \mathit{Att}^*, (\text{'control node'}, ...)...... >$

$\mathit{IEo}7 :=< \text{'PriceIndication'}, \mathit{Att}^*, \text{'int'}, ... >$

Activity *A'4* has the label name *'Validate Gold Price'*, and two properties, *'system task'*, assigned from the property-domain `type`, and *50*, assigned from property-domain `cost`. The former indicates that it is executed by a system functionality and the latter is the estimated cost of implementing this functionality as service operation. Because *A'4* is a system task, it can follow and be followed by only exactly one node. The two relations associated with the property *'system task'* imply this restriction. According to the to-be process of our case study in Section 3 (Fig. 3), *A'4* is described to follow *D'3* and be followed by *M'2*. Among many other relations, we describe here only another relation *'is input of'* of *A'4*. This relation yields that *A'4* takes the Information Entity *'PriceIndication'* as input, which is the price in integer submitted by the customer in the order.

In **step 2** of GAMBUSE, the instantiated to-be activities are grouped into functional SESE fragments using the technique described in [23]. Our case study in Fig. 3 also visualizes graphically the results of this step, decomposing the to-be process into SESE functional fragments that are demarcated by red dashed lines. Notably, each individual activity is shaped as an atomic, non-decomposable SESE functional fragment while the entire process is the coarsest-grained SESE fragment. Instances of the concept `functional fragment` in the SMM are created for the new SESE fragments and added to the SSS. Then the meta-model operators are applied for subsequently grouping the to-be activities into these new fragments[6]. The resulting SESE fragments are a preparatory step identifying only the candidate business services for the next step 3. Next section will explain this step, in which GAMBUSE leverages the same meta-model operators to dissect or recompose the fragments to yield the final business services, while ensuring the right level of granularity of them.

5 Distilling Business Services from the To-Be Process Model

During **step 3** of GAMBUSE, the identified SESE functional fragments[7] in the to-be process are further analyzed to yield the final business services. Starting from these SESE fragments (fragments for short), which are the instances of the SMM concept `functional fragment` and the elements of the to-be SSS, this step leverages a toolkit of several meta-model operators defined in Section 5.1 to subsequently dissect and recompose these fragments in order to distil meaningful to-be business services. Moreover, the resulting business services must comply with essential quality criteria, including high cohesion and loose coupling. Section 5.2 introduces these foundational SOA design criteria for business service design. Lastly, Section 5.3 explains the detail of step 3 of GAMBUSE, grouping the process activities into highly cohesive and loosely coupled business services.

5.1 Meta-Model Operators for Distilling Business Services

From the business perspective it is preferable to identify and distil coarse-grained and highly cohesive business services that collectively shape a self-contained business process. The *Unify operator* facilitates the amalgamation of fragments that exhibit similar functionalities to coarser-grained ones. This operator thus facilitates the reconciliation of the to-be model as far as the granularity of their constituents is concerned. Principally, two unification variants are supported. Firstly, two fragments can be merged in a new coarser-grained one, or secondly, a fragment can be injected into another one.

Definition 5-1. ***Unify operators****, $(F_A \mid F_{2'}) = Unify\ (F_1\ ,\ F_2)$, are composite operators that use two fragments as inputs, and result in a new aggregated one or a new version of an existing one. Inspired by [12], the unification operator utilizes the GraphMerge algorithm [6], which basically consists of four subsequent steps: (1) transform the fragments F_1, F_2 into two directed graphs g_1, g_2, (2) reconcile the labels l_m and l_n of all sub-fragments f_m and f_n ($f_m \in F_1$ and $f_n \in F_2$) (3) fuse two graphs, (4) make resulting graph well-formed respecting the meta-model underpinning (F_1, F_2), i.e., the SMM in this case. The result is a new fragment F_A, which is again instantiated from the SMM and added into the to-be SSS, or a modified version $F_{2'}$ of F_2 in the to-be SSS.*

[6] We do not describe this step in detail because the meta-model operators used for grouping in this step are similar to the ones that will be used in the next step 3. We refer interested readers of the SESE decomposition technique to the work reported in [23].

[7] Note that each individual activity is also a finest-grained SESE fragment.

Extracting a fragment out of a process model is facilitated by the ***extract operator***. This operator simply removes all relations between this fragment and other fragments in the SSS. The other way around, a new fragment can be weaved into a process model with the ***insert operator***, by adding new relations with the existing fragments in the SSS.

5.2 Cohesion and Coupling Criteria

Cohesion and coupling are key guiding principles to ascertain reasonable quality of the resulting business services. Researchers recently studied the use of existing cohesion and coupling criteria for software systems for process design [8]. The work reported in [18,22] provides a comprehensive state-of-the-art in cohesion and coupling metrics that are particularly useful for measuring the quality of different design options of an entire business process. Although these metrics are useful in their own right, they have very limited value to business service engineers if not augmented with additional information for business service identification.

GAMBUSE considers *cohesion* as the degree of the strength of functional relatedness of activities within each business service. Coarse-grained and highly cohesive business services should be identified and distilled from the to-be process for maintainability and reusability purposes. GAMBUSE follows a set of guidelines to ensure a high cohesion degree of the resulting business services (adapted from [17]):

- *Functional cohesion*: A functionally cohesive business service should perform activities that are related to one and only one well-defined business function.
- *Data cohesion*: A data-cohesive business service is the one whose activities have high overlaps of input and output data. Data-cohesive business services are cleanly decoupled from other business services as their activities are hardly related to other activities.
- *Logical cohesion*: A logically cohesive business service is the one whose activities perform a set of independent but logically similar functions (alternatives) of the same general category and are tied together by means of control flows.

In our case study in Section 3 (Fig.3 and Fig. 4), *A'1-Register Order* and *A'2-Verify Order* can be considered to be functionally cohesive, contributing to the same business function “Order Management”. Data cohesion can be checked by computing the overlaps of input/output Information Entities between two activities. However, overlaps in output data are excluded because they reflect different variants that result in the same outcome. For instance, *A'7-Check Availability* and *A'8-Reserve Inventory* are data-cohesive, since they both consume the same input Information Entities: *ItemID(o5) [1...N] and Quantity(o6) [1...N]*. An activity and its extensions can be considered to be logically cohesive, e.g., *A'5-“Validate Price”* and *A'4-“Validate Gold Price”*, since they are two alternatives of the same logical function.

In addition to cohesion, coupling is another criterion for a high-quality process design. Coupling must be analyzed from the global process design perspective to evaluate the inter-dependency grade between the process elements and hence yield the process complexity. It is hypothesized that a process with high coupling grade (high complexity) contains more errors than with low coupling grade [7]. Step 3 of GAMBUSE aims to identify not only highly cohesive but also loosely coupled business services. Loose coupling can help increase the understandability of the process design and isolate a business service from changes of the other, therefore increase the maintainability and reusability of the overall process and also its underlying business services.

5.3 Grouping Process Activities into Business Services

In step 3, GAMBUSE leverages cohesion and coupling criteria in a novel way for mining business services from the SESE fragments in the to-be process model. Particularly, mining is organized in two steps:

- Cohesion criteria assist in deciding whether two fragments should be merged into a new, more cohesive, and higher-order fragment. Our inside-out approach begins with the finest-grained fragments in a model, namely the process activities, and incrementally groups them into a coarser-grained fragment, if they are highly cohesive according to the criteria above. Inside a parent fragment *F*, grouping among child fragments will be iteratively performed until the fix point where no new fragment can be further created. Then, *F* will be neglected, i.e., removed from the to-be SSS, and the new constituted fragments inside *F* will become again the child fragments of the parent fragment of *F* for the next grouping round.

  For instance in our case study in Section 3 (Fig. 3), *A'7* and *A'8* are cohesive and can be unified. Hence, the whole SESE fragment *F1* becomes a cohesive fragment. However, it is not always the case that an SESE fragment becomes a cohesive fragment. Since *A'5* and *A'6* are not cohesive enough, they remain separated. *F3* will then disappear and *A'5* and *A'6* will be further considered for grouping inside *F2*. The other way around, a resulting cohesive fragment might not always be an SESE fragment, e.g., continuing our example inside *F2*, *A'5* is now highly cohesive with *A'4* and can be unified to a new cohesive but non-SESE fragment.
- In the second step of mining, the resulting cohesive fragments will be further analyzed and reinforced according to the coupling criterion. It could be the case that two non-cohesive fragments have too many dependencies and hence still need to be unified into a new fragment. In general, it is hard to automatically detect a local constellation of highly coupled fragments inside the to-be process. Rather, this step requires human involvement from business process analysts who may suggest the grouping alternatives of process elements to reduce the total coupling grade of the process. A coupling metric is crucial in this step to provide the means for evaluating each grouping alternative. Recent work in [7,18,22] introduces a metric that evaluates the coupling grade between all process steps in a model, taking into account also the impact of the coupling degree of different connectors (AND, OR, XOR). This metric could help business service engineers decide whether a grouping alternative can help reduce the total coupling grade of the to-be process.

In conclusion, starting with the SESE functional fragments, this step subsequently groups two fragments *F1* and *F2* into a new one *F*, if they are highly cohesive or highly dependent on each other. Instances of all the newly resulting fragments are created from the SMM and added to the to-be SSS. Meta-model operators needed in this case are: (1) *extract(SSS, F1)*, i.e., extracting the *F1* out of the to-be SSS, (2) *extract(SSS, F2)*, (3) *F=Unify(F1, F2)*, i.e., unifying *F1* and *F2* to *F*, and lastly, (4) *insert(SSS, F)*, i.e., inserting the new *F* to the to-be *SSS*.

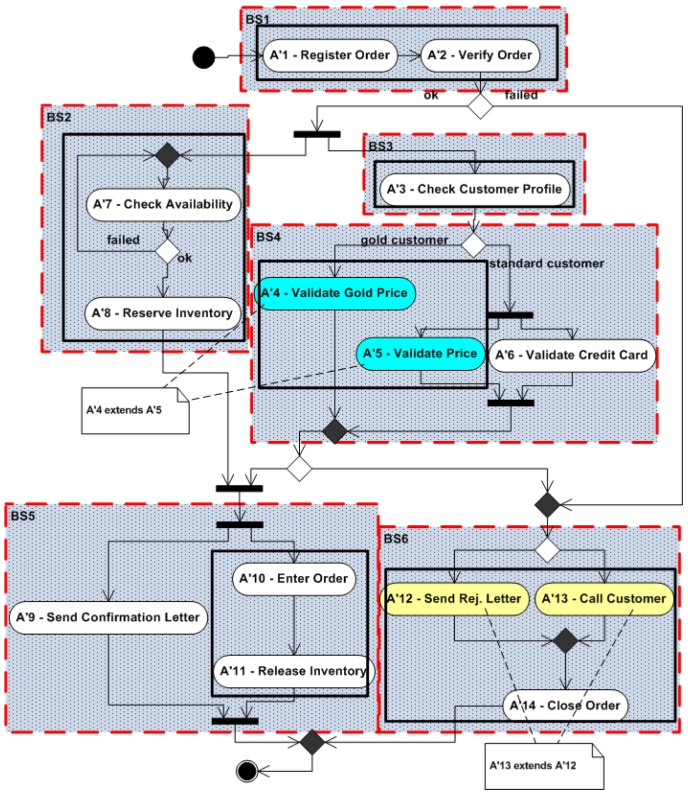


Fig. 7. To-Be Process with final business services based on cohesion and coupling criteria

At the end of this step, the resulting fragments embody the "final" business services that should be included in the to-be SSS. Business services can be instantiated from the SMM and added to the to-be SSS as follows: For each final fragment *F* in the to-be SSS, which is actually an instance of the `functional fragment` in the SMM, (1) create a new instance *S* of the `Service` in the SMM with type `"Business Service"` and add it to the to-be SSS, (2) link the service *S* to the fragment *F* with the relation `"executes"`, indicating that *F* is executed by *S*.

Fig. 7 depicts the final business services resulting from cohesion- and coupling-based grouping of activities. The resulting fragments after the first step of grouping based only on the cohesion criteria are demarcated by the black lines. In the second step, more model elements are grouped together to reduce the total coupling grade of the process, resulting in the final business services demarcated by the red dashed lines. For instance, *A'6* is included in *BS4* and *A'9* is included in *BS5*, although *BS4* and *BS5* then become not as cohesive as before. Although further groupings are still able to reduce the total coupling grade, the business service engineers decide to stop here because otherwise, the resulting business services are too coarse-grained and no longer sufficiently cohesive. They also believe that they have reached a reasonable design quality of the to-be process.

6 Detecting and Assessing the Reusability of As-Is Systems

Functional overlaps and discrepancies, as well as misfits in behaviour and policy between the as-is and to-be process are detected in the **step 4** of GAMBUSE, aiming to assess the reusability of existing software functionalities that currently perform specific activities in the as-is process. Given the formal Service Schema Specifications (SSS) of the as-is and to-be process derived from step 1, several activity matching approaches are introduced in this section to detect the equivalence between an as-is and a to-be activity. These comprise of:

- Discovery of functional overlaps or extensions based on simple label matching.
- Syntactic matching of activity signatures.
- Behavioural matching of activity signatures.

All these three matching approaches are performed by the meta-model operator *Match* which operates on the SSS as-is and to-be activities and other related SSS elements such as `Information Entities`, `Operation Conditions`, etc.

Firstly, functional overlaps between as-is and to-be activities in GAMBUSE rely on simple label matching, since we assume that activities always have unique and meaningful labels regardless of their granularity level. Furthermore in the to-be process, we also assume that activities and their extensions are marked by the business analysts. Our approach is simple, if a to-be activity *A'* is a functional overlap of an as-is activity *A*, all extended activities of *A'* will be functional extensions of *A*. For instance in our case study in Section 3 (Fig. 3), *A'5 ("Validate Price")* and *A4 ("Validate Price")* have the same label and are functionally overlap. Hence, *A'4 ("Validate Gold Price")* (the extension of *A'5*) can be considered as a functional extension of *A4*.

In addition to functional overlaps and extensions based on label matching, we require the syntactic and behavioural signature compatibility of as-is and to-be activities. Our work so far concentrates on matching the signatures of activities.

Definition 6-1: ***Activity Signature****. The signature of an activity A is defined as a 5-tuple A = (inputs, outputs, preconditions, post-conditions, invariants) with*

inputs = (Information Entity), outputs = (Information Entity*),*

preconditions = (Operation Condition), post-conditions = (Operation Condition*), invariants = (Operation Condition*), and*

Operation Condition = (Constraint).*

Information Entity, Operation Condition and Constraint are related elements of the element Activity in the SSS. Inputs and outputs signify the <u>syntactic signature</u> and pre-, post-conditions and invariants signify the <u>behavioural signature</u>.

Example 6-1: In our case study in Section 3 (Fig. 3 and Fig. 4), the signature of the to-be activity *A'4 ("Validate Gold Price")* contains:
inputs=< (ItemID(o5), PriceIndication(o7) [1…N] , customerType(c2)>
outputs=< verificationResult(o9), status(o3) >,
preconditions=<"customerType='gold'">,
post-conditions=<"status='PriceValidated'">, invariants=<>

and the signature of the as-is activity *A4 ("Validate Price")* contains
inputs=< (ItemID(o5), PriceIndication(o7) [1...N] >,
outputs=< verificationResult(o9), status(o3) >,
preconditions=<>,
post-conditions=<"status='PriceValidated'">, invariants=<>

Signature matching comprises of two sub-steps: syntactic signature matching with inputs and outputs (Section 6.1) and behavioural signature matching with preconditions, post-conditions and invariants (Section 6.2). Lastly, Section 6.3 introduces the meta-model operators *Match* to yield the activity equivalence and *Disparity* to detect the activities in the to-be process that cannot be matched.

6.1 Syntactic Signature Matching

Syntactic signature matching concentrates on the matching of input/output data types of as-is and to-be activities. Data in our approach are captured in Information Entities, which denote the elements in the SSS and have defined XSD types. The following theoretical approach of syntactic signature matching is based on the work reported in [16,20,21] which leverages type checking technique for checking version compatibility of service signature interfaces:

Definition 6-2: ***Contra-Variance of Input Information Entities.*** *Let IN be the collection of input Information Entities ie of an activity A and let IN' be the collection of input Information Entities ie' of an activity A'. Then IN is contra-variant to IN', iff $\forall ie \in IN$, $\exists ie' \in IN'$, so that ie' has the same or extended XSD type*[8] *as ie.*

Definition 6-3: ***Co-Variance of Output Information Entities.*** *Let Out be the collection of output Information Entities ie of an activity A and let Out' be the collection of output Information Entities ie' of an activity A'. Then Out is co-variant to Out', iff $\forall ie' \in Out'$, $\exists ie \in Out$, so that ie has the same or extended XSD type as ie'.*

Definition 6-4: ***Syntactic signature compatibility [20].*** *A to-be activity A_ω, has a syntactically compatible signature with A_α, denoted by $A_\alpha \oplus A_\omega$, iff inputs of A_α are contra-variant to inputs of A_ω(see def 6-2) and outputs of A_α are co-variant to outputs of A_ω (see def 6-3)*

Example 6-2: Continuing the example 6-1, we claim that the to-be activity *A'4* has a syntactically compatible signature with the as-is activity *A4* because:

- In our case study in Section 3, as-is and to-be processes use the common Information Entities with primitive types. Only in the to-be process, some extra Information Entities are newly introduced.
- *A'4* consumes one input Information Entity more, i.e., the *customerType (c2).*
- *A'4* produces the same output Information Entities as *A4*.

[8] An XSD type extension can be a primitive type extension, e.g., long extends int, or a complex type extension, i.e., $\forall e$ - nested Information Entities in *ie*, $\exists e'$ - nested Information Entities in *ie'*, so that *e'* has the same or extended XSD type as *e*.

The next step is to consider the behavioural signature compatibility between an as-is and a to-be activity.

6.2 Behavioural Signature Matching

The behavioural signature of an activity is defined as a set of preconditions, post-conditions and invariants that constrain the activity execution[9]. Global business rules and policy that constrain all (or a subset of) activities may be translated into local rules of each activity and expressed by a set of `Operation Conditions` in the SSS, namely the `Pre-, Post-conditions and Invariants`. Each of these `Operation Conditions` consists of a number of `Constraints.` In the current version of GAMBUSE, a `Constraint` *ct* is expressed by restricting the properties `"value range"` R_i of one or many SSS `Information Entities` ie_i with primitive XSD types, denoted by $ct \rightarrow \sum_i (ie_i, R_i)$, $i=1...N$.

Example 6-3: For instance in our case study in Section 3 (Fig. 3 and Fig. 4), let *vR* be the property "*value range*" of the Information Entity *orderDelay(o8),* and *vR* is initially set to [1…30] (days) by default by the company AutoScore, meaning that in general case an order can delay maximum 30 days. Then, the precondition *"Order Delay < 1 week"* of *A'9* can be translated into *vR = [1…7] (days).*

Definition 6-5: ***Constraint Extension*****:** *A constraint ct is an extension of another constraint ct', denoted by* $ct \supseteq ct'$, *iff*

- *both of them can be translated into a set of restricted value ranges of Information Entities, i.e.,* $ct \rightarrow \sum_i (ie_i, R_i)$, $i=1...N$ *and* $ct' \rightarrow \sum_j (ie'_j, R_j)$ $j=1...M$, *and*
- $\forall ie'_j$, $\exists ie_i$, *so that* ie_i *has the same or extended primitive XSD type as* ie'_j , *and* $R_i \supseteq R_j$.

Example 6-4: Continuing the example 6-3, a sample extension vR_1 of the precondition *vR* of *A'9* could be "*Order Delay < 2 week*", i.e., $vR_1 = [1...14]$ *(days).*

Definition 6-6: ***Contra-Variance of Preconditions.*** *Let Pre be the collection of precondition constraints of an activity A and let Pre' be the collection of precondition constraints of another activity A'. Then Pre is contra-variant to Pre', iff* $\forall ct \in Pre$, $\exists ct' \in Pre'$, *so that* $ct \subseteq ct'$ *(see def 6-5).*

Definition 6-7: ***Co-Variance of Post-conditions.*** *Let Post be the collection of post-condition constraints of an activity A and let Post' be the collection of post-condition constraints of another activity A'. Then Post is co-variant to Post', iff* $\forall ct' \in Post'$, $\exists ct \in Post$, *so that* $ct' \subseteq ct$ *(see def 6-5).*

Definition 6-8: ***Contra-Variance of Invariants****. Let Inv be the collection of invariant constraints on an activity A and let Inv' be the collection of invariant constraints on another activity A'. Then Inv is contra-variant to Inv', iff* $\forall ct \in Inv$, $\exists ct' \in Inv'$, *so that* $ct \subseteq ct'$ *(see def 6-5).*

[9] In fact, the behaviour of an activity with pre-, post-conditions and invariants implicitly reflects its functionality. Hence, behavioural signature matching may be considered to complement our simple functional matching approach based on label matching mentioned before.

We are now able to derive the definition of behavioural signature compatibility.

Definition 6-9: ***Behavioural signature compatibility*** *To-be activity A_ω has behaviourally compatible signature with as-is activity A_α, denoted by $A_\alpha \approx A_\omega$, iff*

- *The preconditions and invariants of A_α are contra-variant to the preconditions and invariants of A_ω (see def 6-6 and def 6-8) and*
- *The post-conditions of A_α are co-variant to the post-conditions of A_ω (see def 6-7).*

Example 6-5: Continuing the examples 6-1 and 6-2 regarding our case study in Section 3 (Fig. 3), we claim that the to-be *A'4* has a behaviourally compatible signature with the as-is *A4* since:

- *A'4* requires the precondition *"customerType=gold"* while *A4* has no precondition.
- Both *A'4* and *A4* require the post-condition *"status=PriceValidated"*.
- Both *A'4* and *A4* have no invariant.

6.3 Match and Disparity Operators

Given the theoretical approaches for functional overlap (or extension) and (syntactic and behavioural) signature compatibility, we define the meta-model operator *Match* to yield activity equivalence.

Definition 6-10: ***Match operator.*** *This operator works on the SSS elements, i.e., an as-is activity A_α and a to-be activity A_ω and other related elements, to yield equivalence relation '$\equiv$' between these two activities, denoted by $A_\alpha \equiv A_\omega$, meaning that A_ω can be substituted by A_α. We define $A_\alpha \equiv A_\omega$ iff*

- *A_ω is a functional overlap or extension of A_α and*
- *$A_\alpha \oplus A_\omega$ (see def 6-4) and $A_\alpha \approx A_\omega$ (see def 6-9).*

In addition, the equivalence relation detected by the *Match* operator is distinguished between ***absolute*** and ***partial*** equivalence. In particular, A_α is absolutely equivalent to A_ω if it is a functional overlap of A_ω and has exactly the same syntactic and behavioural signature as A_ω. That means the as-is functionalities implementing A_α can be completely repurposed to implement A_ω. All other cases are partial equivalence, meaning that the as-is functionalities must be revamped before being reused.

Beside activity equivalence, disparities between as-is and to-be process can be identified by the *disparity operator*. This operator is basically a variant of the FullOuterMatch introduced in [4], revealing activities that appear in the to-be model but cannot be matched to any activities in the as-is model, but not the other way around.

Definition 6-11: ***Disparity operator****. The disparity operator takes the as-is model M_α, and the to-be model M_ω as inputs, and returns process activities that appear in the to-be but not in the as-is model, i.e., new activities that were added to or revamped in the to-be model. Hence, this operator returns all $A_\omega \in M_\omega$, where $\neg(A_\alpha \equiv A_\omega)$, $\forall A_\alpha \in M_\alpha$.*

Example 6-6: Regarding our case study in Section 3 (Fig. 3), activity matching yields:

- Absolute equivalence: $A_2 \equiv A'_2$, $A_4 \equiv A'_5$, $A_3 \equiv A'_6$, $A_{5\text{-}4} \equiv A'_{11}$, $A_{6\text{-}1} \equiv A'_{12}$, $A_{6\text{-}2} \equiv A'_{14}$.
- Partial equivalence: $A_1 \equiv A'_1$, $A_4 \equiv A'_4$, $A_{5\text{-}3} \equiv A'_7$, $A_{5\text{-}2} \equiv A'_9$, $A_{5\text{-}1} \equiv A'_{10}$, $A_{6\text{-}1} \equiv A'_{13}$.
- Disparities: A'_3, A'_8.

At the end of this matching step, the business service engineers are provided with:

- Final business services collectively constituting the to-be process.
- Reusability of as-is process activities for the to-be process with associated information like which existing software functionalities of which existing systems are currently implementing these as-is activities.

In step 5 of GAMBUSE, with this information business service engineers can make educated decisions on the realization strategy for each newly conceived business service. For instance, revamping or repurposing existing functionalities will be applied for business services with high reusability of as-is software assets that can be efficiently maintained and migrated to SOA environment. In contrast, business services with low reuse or with reuse of cost-intensive functionalities will be considered to be newly developed in-house, outsourced, or rent from external service providers.

Example 6-7: Fig. 7 depicts the final business services in the to-be process of our case study. In addition, the matching results given in example 6-6 yield the reusability of as-is functionalities. In step 5 of GAMBUSE, the initial decisions of the business service engineers are as follows:

- Reusing existing systems to implement BS_1, BS_4, BS_5, BS_6 since these business services contain only matched activities that can be realized by repurposing or revamping existing as-is functionalities.
- Developing BS_2 and BS_3 from scratch, since they have no or low reuse of existing functionalities.

However, the cost of modifying the existing rigid CRM and ERP systems for implementing BS_5 and BS_6 and the cost of migrating them into SOA environment are considerably very high. Hence, the business service engineers finally change their minds to purchase new SOA-based enterprise packages for realizing BS_5 and BS_6.

7 Conclusions and Further Research Work

In this article, we have introduced the novel business service engineering methodology, GAMBUSE, for developing SOA-based applications. GAMBUSE provides the techniques to identify modular, cohesive, and reusable business services in a redesigned to-be process, while taking into account also the reuse of existing functionalities exposed by existing software assets. In contrast to existing service and gap analysis techniques, the methodology that we presented here represents a first attempt in assisting business service engineers to identify reusable parts of the business services in as-is process models, stressing reliance on a reference service meta-models, and taking into account a set of development strategies and foundational SOA design principles.

The research results presented in this article are core in nature. Improvements are needed in several directions. Firstly, we would like to further formalize the semantics of the meta-model operators, e.g., defining the matching operator so that it can cope with business protocol- and transactional semantics. Then, we intend to develop an experimental prototype to further study the ramifications of the operators. Our current implementation uses ConceptBase, a deductive database management system, within which we have defined the operators as a set of declarative and active rules that operate on as-is and to-be model and meta-model facts; however, this approach is too restrictive for our purposes. Furthermore, we may extend GAMBUSE with modelling constructs for capturing business value, so candidate business services may not only be identified and developed relying on both technical functional and non-functional criteria, but also on the degree in which they contribute to business objectives. Last but not least, we also foresee the necessity to incorporate with industrial partners to gain experience and lesions learned from applying the methodology in the industry.

References

1. Andrikopoulos, V., Benbernou, S., Papazoglou, M.P.: Managing the evolution of service specifications. In: Bellahsène, Z., Léonard, M. (eds.) CAiSE 2008. LNCS, vol. 5074, pp. 359–374. Springer, Heidelberg (2008)
2. Armstrong, C., Underbakke, B.: MDA Legacy Modernization Case Study: State of Wisconsin Unemployment Insurance Division, Architecture-Driven Modernization Workshop - A Model-driven Approach to Modernizing IT Systems, Chicago, IL, USA (2004)
3. Bernstein, P.A., Melnik, S.: Model management 2.0: manipulating richer mappings. In: Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data, ACM, NY (2007)
4. Bernstein, P., Halevy, A., Pottinger, R.: A vision for management of complex models. ACM SIGMOD Record 29(4), 55–63 (2000)
5. Bolstorff, P., Rosenbaum, R.: Supply Chain Excellence: A Handbook for Dramatic Improvement Using the Scor Model, 2nd edn. Ed. AMACOM (2007)
6. Buneman, P., et al.: Theoretical Aspects of Schema Merging. In: Pirotte, A., Delobel, C., Gottlob, G. (eds.) EDBT 1992. LNCS, vol. 580, pp. 152–167. Springer, Heidelberg (1992)
7. Cardoso, J., Vanderfeesten, I., Reijers, H.A.: A weighted coupling metric for business process models. In: Krogstie, J., Opdahl, A.L., Sindre, G. (eds.) CAiSE 2007 and WES 2007. LNCS, vol. 4495, Springer, Heidelberg (2007)
8. Guceglioglu, A.S., Demirors, O.: Using Software Quality Characteristics to Measure Business Process Quality. In: van der Aalst, W.M.P., Benatallah, B., Casati, F., Curbera, F. (eds.) BPM 2005. LNCS, vol. 3649, pp. 374–379. Springer, Heidelberg (2005)
9. Jeston, J., Nelis, J.: Business Process Management: Practical Guidelines to Successful Implementations, 3rd edn. Butterworth-Heinemann (2006)
10. Mantell, K.: From UML to BPEL: MDA in a Web Services World, DeveloperWorks (September 2005)
11. Marks, E., Bell, M.: Service Oriented Architecture: A planning and implementation guide for Business and Technology. John Wiley & Sons, Chichester (2006)
12. Melnik, S. (ed.): Generic Model Management. LNCS, vol. 2967. Springer, Heidelberg (2004)
13. Miller, J., Mukerji, J.: MDA Guide Version 1.0.1, Document number omg/2003-06-01 (2003), `http://www.omg.com/mda`

14. Palmer, N., Mooney, L.: Building a business case for BPM – a fast path to real result. White paper (2007)
15. Papazoglou, M.P., van den Heuvel, W.J.: Business Process Development Lifecycle Methodology. Communications of ACM (October 2007)
16. Papazoglou, M.P.: The Challenges of Service Evolution. In: Bellahsène, Z., Léonard, M. (eds.) CAiSE 2008. LNCS, vol. 5074, pp. 1–15. Springer, Heidelberg (2008)
17. Papazoglou, M.: Web service: principle and technology. Pearson Prentice Hall (2008)
18. Reijers, H.A., Vanderfeesten, I.T.P.: Cohesion and Coupling Metrics for Workflow Process Design. In: Desel, J., Pernici, B., Weske, M. (eds.) BPM 2004. LNCS, vol. 3080, pp. 290–305. Springer, Heidelberg (2004)
19. Ulrich, W.: Aligning Existing IT Architectures with SOA, OMG SOA Information Day (2004), `http://soa.omg.org/SOA-Info-Day_12-06.htm`
20. Van den Heuvel, W.J.: Aligning Modern Business Processes and Legacy Applications. MIT Press, Cambridge (2007)
21. Van den Heuvel, W.J.: Matching and Adaptation: Core Techniques for MDA-(ADM)-driven Integration of new Business Applications with Wrapped Legacy Systems. In: MELS Workshop, IEEE, Los Alamitos (2004)
22. Vanderfeesten, I., Reijers, H.A., van der Aalst, W.M.P.: Evaluating Workflow Process Designs using Cohesion and Coupling Metrics. Computers in Industry 59(5), 420–437 (2008)
23. Vanhatalo, J., Völzer, H., Leymann, F.: Faster and more focused control-flow analysis for business process models through SESE decomposition. In: Krämer, B.J., Lin, K.-J., Narasimhan, P. (eds.) ICSOC 2007. LNCS, vol. 4749, pp. 43–55. Springer, Heidelberg (2007)
24. Watson, A.: A Brief History of MDA, Upgrade. The European Journal for the Informatics Profesional IX(2) (2008)

Appendix A: Generic Meta-model Operators in GAMBUSE

Primitives	
Signature	**Description**
c = extract (M,c)	Returns classifier(s) c in model M.
M' = remove (M,c)	Returns a sub-model of model M that does not contain classifiers in c.
M' = insert (M,c)	Returns a "super-model" of model M that also contains classifier c.
Higher-order operators	
Signature	**Description**
$(c_\alpha \equiv c_\omega)$= Match$_\Gamma(M_\alpha, M_\omega)$	Returns pairs of equivalent classifiers from the as-is model, M_α, and the to be model M_ω according to equivalence level Γ.
$(c_\Lambda \mid c_{2'})$ = Unify (c_1, c_2)	Returns the new classifier C_Λ that composes two classifiers C_1, C_2. Alternatively, it yields a new version of $C_{2'}$ that subsumes C_1.
(c_1, c_2)= Decompose(C_Λ)	Returns the two new classifiers c_1, c_2 that are decomposed from C_Λ
c_ω Disparity (M_α, M_ω)	Returns all classifiers c_ω from the to-be model M_ω that are not in the as-is model M_α.
$M_{\omega'}$ Subtract (M_ω, c_ω) or Scope(M_ω, c_ω)	Returns the scoped model $M_{\omega'}$ without c_ω

Web Service Composition via the Customization of Golog Programs with User Preferences[*]

Shirin Sohrabi, Nataliya Prokoshyna, and Sheila A. McIlraith

Department of Computer Science, University of Toronto, Toronto, Canada
{shirin,nataliya,sheila}@cs.toronto.edu

Abstract. We claim that user preferences are a key component of effective Web service composition, and one that has largely been ignored. In this paper we propose a means of specifying and intergrating user preferences into Web service composition. To this end, we propose a means of performing automated Web service composition by exploiting a flexible template of the composition in the form of a generic procedure. This template is augmented by a rich specification of user preferences that guide the instantiation of the template. We exploit the agent programming language Golog to represent our templates as Golog generic procedures and we exploit a first-order preference language to represent rich qualitative temporally-extended user preferences. From these we generate Web service compositions that realize a given generic procedure, satisfying the user's hard constraints and optimizing for the user's preferences. We prove our approach is sound and optimal. Our system, GologPref, is implemented and interacting with services on the Web. The language and techniques proposed in this paper can be integrated into a variety of approaches to Web or Grid service composition.

Preamble

We were inspired to include the research that follows in this volume in honour of John Mylopoulos because it touches upon at least two different themes that John has addressed in his research in recent years. In particular, John's work on Tropos has focused on the specification of information system requirements in terms of actors, goals, and interdependencies. The Tropos methodology can be realized in a variety of agent programming environments, including variants of Golog. John and his colleagues have applied the Tropos methodology to the design of a variety of software systems, including the design of Web services. In this context our Golog Web service composition templates can be seen as a specification of the requirements of our Web service composition, while our user preferences correspond to a specification of soft requirements to be optimized.

[*] An earlier version of this paper originally appeared as *Web Service Composition via Generic Procedures and Customizing User Preferences* in [1] and is reprinted in revised form with the permission of the publishers.

1 Introduction

Web services provide a standardized means for diverse, distributed software applications to be published on the Web and to interoperate seamlessly. Simple Web accessible programs are described using machine-processable descriptions and can be loosely composed together to achieve complex behaviour. The weather service at www.weather.com and the flight-booking services at www.aircanada.ca, are examples of Web applications that can be described and composed as Web services. They might be coupled as part of a travel-booking service, for example.

Automated Web service composition is one of many interesting challenges facing the Semantic Web. Given computer-interpretable descriptions of: the task to be performed, the properties and capabilities of available Web services, and possibly some information about the client or user's specific constraints, *automated Web service composition* requires a computer program to automatically select, integrate and invoke multiple Web services in order to achieve the specified task in accordance with any user-specific constraints. Compositions of Web or Grid services are necessary for realizing both routine and complex tasks on the Web (resp. Grid) without the need for time-consuming manual composition and integration of information. Compositions are also a useful way of enforcing business rules and policies in both Web and Grid computing.

Fully automated Web service composition has been characterized as akin to both an artificial intelligence (AI) planning task and to a restricted software synthesis task (e.g., [2]). A composition can be achieved using classical AI planning techniques by conceiving services as primitive or complex actions and the task description specified as a (final state) goal (e.g., [3,4]). This approach has its drawbacks when dealing with data. In general, the search space for a composition (aka plan) is huge because of the large number of available services (actions), which grow far larger with grounding for data.

A reasonable middle ground which we originally proposed in [5,2] is to specify a flexible template of the composition in the form of a *generic procedure* and to customize such a procedure with *user constraints*. We argued that many of the tasks performed on the Web or on intranets are repeated routinely, and the basic steps to achieving these tasks are well understood, at least at an abstract level – travel planning is one such example. Nevertheless, the realization of such tasks varies as it is tailored to individual users. As such, our proposal was to specify such tasks using a workflow template or generic procedure and to customize the procedure with user constraints at run time. Such an approach is generally of the same complexity as planning but the search space is greatly reduced, and as such significantly more efficient than planning without such generic advice.

In [2] we proposed to use an augmented version of the agent programming language Golog [6] to specify our generic procedures or workflows with sufficient nondeterminism to allow for customization. (E.g., *"book inter-city transportation, local transportation and accommodations in any order"*). User constraints (e.g., *"I want to fly with Air Canada."*) were limited to hard constraints (as opposed to "soft"), were specified in first-order logic (FOL), and were applied to the generic procedure at run-time to generate a user-specific composition of services. A similar approach was adopted using hierarchical task networks (HTNs) to represent generic procedures or templates, and realized using the HTN planner, SHOP2 (e.g., [7]) without user customization of the HTN template.

In this paper, we extend our Golog framework for Web service composition, customizing Golog generic procedures not only with hard constraints but with *soft* user constraints (henceforth referred to as *preferences*). These preferences are defeasible and may not be mutually achievable. We argue that user preferences are a critical and missing component of most existing approaches to Web service composition. User preferences are key for at least two reasons. First, the user's task (specified as a goal and/or generic procedure with user constraints) is often under constrained. As such, it induces a family of solutions. User preferences enable a user to specify properties of solutions that make them more or less desirable. The composition system can use these to generate preferred solutions.

A second reason why user preferences are critical to Web service composition is with respect to *how* the composition is performed. A key component of Web service composition is the selection of specific services used to realize the composition. In AI planning, primitive actions (the analogue of services) are selected for composition based on their preconditions and effects, and there is often only one primitive action that realizes a particular effect. Like actions, services are selected for composition based on functional properties such as inputs, output, preconditions and effects, but they are also selected based on domain-specific nonfunctional properties such as, in the case of airline ticket booking, whether they book flights with a carrier the user prefers, what credit cards they accept, how trusted they are, etc. By integrating user preferences into Web service composition, preferences over services (the *how*) can be specified and considered along side preferences over the solutions (the *what*).

In this paper we recast the problem of Web service composition as the task of finding a composition of services that achieves the task description (specified as a generic procedure in Golog), that achieves the user's hard constraints, and that is *optimal* with respect to the user's preferences. To specify user preferences, we exploit a rich qualitative preference language, based on the $\mathcal{LPP}$ language proposed by Bienvenu et al. to specify users' preferences in a variant of linear temporal logic (LTL) [8,9]. We prove the soundness of our approach and the optimality of our compositions with respect to the user's preferences. Our system can be used to select the optimal solution from among families of solutions that achieve the user's stated objective. Our system is implemented in Prolog and integrated with a selection of scraped Web services that are appropriate to our test domain of travel planning.

The work presented here is predicated on the assumption that Web services have been described in a computer-interpretable form. This is the starting point for most work on semantic Web services [5] and a great deal of effort has gone into the development of ontologies for precisely this purpose. In this paper, Web service descriptions are presented in FOL, *not* in one of the typical Semantic Web languages such as OWL [10] nor more specifically in terms of a semantic Web service ontology such as OWL-S [11], WSMO [12] or SWSO [13]. Nevertheless, it is of direct significance to semantic Web services. As noted in (e.g., [11]) process models, necessary for Web service composition, cannot be expressed in OWL while preserving all and only the intended interpretations of the process model. OWL (and thus OWL-S) is not sufficiently expressive. Further OWL reasoners are not designed for the type of inference necessary for Web service composition. For both these reasons, Web service composition systems generally translate the relevant aspects of service ontologies such as OWL-S into internal representations such as the Planning Domain Definition Language (PDDL) that are more amenable to AI planning (e.g., [7,14]). Golog served as one of the inspirations for what is now OWL-S [5] and all the OWL-S constructs have translations into Golog [15]. Further, the semantics of the OWL-S process model has been specified in situation calculus [13,16]. Thus, our Golog generic procedures can be expressed in OWL-S and likewise, OWL-S ontologies can be translated into our formalism. We do not have a current implementation of this translation, but it is conceptually straightforward.

2 Situation Calculus and Golog

We use the situation calculus and FOL to describe the functional and nonfunctional properties of our Web services. We use the agent programming language Golog to specify composite Web services and to specify our generic procedures. In this section, we review the essentials of situation calculus and Golog.

The situation calculus is a logical language for specifying and reasoning about dynamical systems [6]. In the situation calculus, the *state* of the world is expressed in terms of functions and relations (fluents) relativized to a particular *situation* s, e.g., $F(\boldsymbol{x}, s)$. In this paper, we distinguish between the set of fluent predicates, $\mathcal{F}$, and the set of non-fluent predicates, $\mathcal{R}$, representing properties that do not change over time. A situation s is a *history* of the primitive actions, $a \in \mathcal{A}$, performed from a distinguished initial situation S_0. The function $do(a, s)$ maps a situation and an action into a new situation thus inducing a tree of situations rooted in S_0. $Poss(a, s)$ is true if action a is possible in situation s.

Web services such as the Web exposed application at www.weather.com are viewed as actions in the situation calculus and are described as actions in terms of a situation calculus basic action theory, $\mathcal{D}$. The details of $\mathcal{D}$ are not essential to this paper but the interested reader is directed to [6,16,2] for further details.

Golog [6] is a high-level logic programming language for the specification and execution of complex actions in dynamical domains. It builds on top of the situation calculus by providing Algol-inspired extralogical constructs for assembling primitive situation calculus actions into complex actions (aka *programs*) δ. These complex actions simply serve as constraints upon the situation tree. Complex action constructs include the following:

nil – the empty program
a – primitive action
$\phi?$ – test action
$\pi x.\,\delta$ – nondeterministic choice of argument
$\delta_1;\delta_2$ – sequences (δ_1 is followed by δ_2)
$\delta_1|\delta_2$ – nondeterministic choice between δ_1 and δ_2
if ϕ **then** δ_1 **else** δ_2 **endif** – conditional
while ϕ **do** δ **endW** – loop
proc $P(\boldsymbol{v})$ δ **endProc** – procedure

We also include the construct **anyorder**$[\delta_1,\ldots,\delta_n]$ which is encoded as the nondeterministic choice of all possible permutaions of the sequencing of $\delta_1,\ldots,\delta_n$. The conditional and while-loop constructs are defined in terms of other constructs. For the purposes of Web service composition we generally treat iteration as finitely bounded by a parameter k. Such finitely bounded programs are called *tree programs.*

$$\textbf{if } \phi \textbf{ then } \delta_1 \textbf{ else } \delta_2 \textbf{ endIf} \overset{\text{def}}{=} [\phi?;\delta_1] \mid [\neg\phi?;\delta_2]$$

$$\textbf{while}_1(\phi)\ \delta\ \textbf{ endWhile} \overset{\text{def}}{=} \textbf{ if } \phi \textbf{ then } \delta \textbf{ endIf}$$ [1]

$$\textbf{while}_k(\phi)\ \delta \textbf{ endWhile} \overset{\text{def}}{=}$$
$$\textbf{if } \phi \textbf{ then } [\delta;\ \textbf{while}\ _{k-1}(\phi)\delta \textbf{ endWhile}] \textbf{ endIf}$$

These constructs can be used to write programs in the language of the domain theory, or more specifically, they can be used to specify both composite Web services and also generic procedures for Web service composition. E.g.[2],

$bookAirTicket(\boldsymbol{x})$; **if** far **then** $bookRentalCar(\boldsymbol{y})$ **else** $bookTaxi(\boldsymbol{y})$ **endIf** $bookRentalCar(\boldsymbol{x})$; $bookHotel(\boldsymbol{y})$.

In order to understand how we modify Golog to incorporate user preferences, the reader must understand the basics of Golog semantics. There are two popular semantics for Golog programs: the original evaluation semantics [6] and a related single-step transition semantics that was proposed for on-line execution of concurrent Golog programs [17]. The transition semantics is axiomatized through two predicates $Trans(\delta, s, \delta', s')$ and $Final(\delta, s)$. Given an action theory $\mathcal{D}$, a program δ and a situation s, *Trans* defines the set of possible successor configurations (δ', s') according to the action theory. *Final* defines whether a program

[1] **if-then-endIf** is the obvious variant of **if-then-else-endIf**.
[2] Following convention we will generally refer to fluents in situation-suppressed form, e.g., $at(Toronto)$ rather than $at(Toronto, s)$. Reintroduction of the situation term is denoted by $[s]$. Variables are universally quantified unless otherwise noted.

successfully terminated, in a given situation. *Trans* and *Final* are defined for every complex action. A few examples follow. (See [17] for details):

$$\begin{aligned}
Trans(nil, s, \delta', s') &\equiv False \\
Trans(a, s, \delta', s') &\equiv Poss(a[s], s) \wedge \delta' = nil \wedge s' = do(a[s], s) \\
Trans(\phi?, s, \delta', s') &\equiv \phi[s] \wedge \delta' = nil \wedge s' = s \\
Trans([\delta_1; \delta_2], s, \delta', s') &\equiv Final(\delta_1, s) \wedge Trans(\delta_2, s, \delta', s') \\
&\vee \exists \delta''.\delta' = (\delta''; \delta_2) \wedge Trans(\delta_1, s, \delta'', s') \\
Trans([\delta_1 \mid \delta_2], s, \delta', s') &\equiv Trans(\delta_1, s, \delta', s') \vee Trans(\delta_2, s, \delta', s') \\
Trans(\pi(x)\delta, s, \delta', s') &\equiv \exists x.Trans(\delta^v_x, s, \delta', s') \\
Final(nil, s) &\equiv \text{TRUE} \\
Final(a, s) &\equiv \text{FALSE} \\
Final([\delta_1; \delta_2], s) &\equiv Final(\delta_1, s) \wedge Final(\delta_2, s)
\end{aligned}$$

Thus, given the program $bookCar(\boldsymbol{x}); bookHotel(\boldsymbol{y})$, if the action $bookCar(\boldsymbol{x})$ is possible in situation s, then

$$Trans([bookCar(\boldsymbol{x}); bookHotel(\boldsymbol{y})], s, bookHotel(\boldsymbol{y}), do(bookCar(\boldsymbol{x}), s))$$

describes the only possible transition according to the action theory. $do(bookCar(\boldsymbol{x}), s)$ is the transition and $bookHotel(\boldsymbol{y})$ is the remaining program to be executed. Using the transitive closure of *Trans*, denoted *Trans**, one can define a *Do* predicate as follows. This *Do* is equivalent to the original evaluation semantics *Do* [17].

$$Do(\delta, s, s') \overset{\text{def}}{=} \exists \delta'.Trans^*(\delta, s, \delta', s') \wedge Final(\delta', s'). \tag{1}$$

Given a domain theory, $\mathcal{D}$ and Golog program δ, program execution must find a sequence of actions $\boldsymbol{a}$ (where $\boldsymbol{a}$ is a vector of actions) such that: $\mathcal{D} \models Do(\delta, S_0, do(\boldsymbol{a}, S_0))$. $Do(\delta, S_0, do(\boldsymbol{a}, S_0))$ denotes that the Golog program δ, starting execution in S_0 will legally terminate in situation $do(\boldsymbol{a}, S_0)$, where $do(\boldsymbol{a}, S_0)$ abbreviates $do(a_n, do(a_{n-1}, \ldots, do(a_1, S_0)))$. Thus, given a generic procedure, described as a Golog program δ, and an initial situation S_0, we would like to infer a terminating situation $do(\boldsymbol{a}, S_0)$ such that the vector $\boldsymbol{a}$ denotes a sequence of Web services that can be performed to realize the generic procedure.

3 Specifying User Preferences

In this section, we describe the syntax of the first-order language we use for specifying user preferences. This description follows the $\mathcal{LPP}$ language we proposed in [8,9] for preference-based planning. The semantics of the language is described in the situation calculus. We provide an informal description here, directing the reader to [8,9] for further details. Our language is richly expressive, enabling the expression of static as well as temporal preferences, and action-centric as well as state-centric preferences. Unlike many preference languages, it induces a total order over the compositions, which avoids the high degree of incomparability experienced by many other non-quantitative preference languages, and simplifies computation of preferred compositions. Our language is qualitative, rather than ordinal or quantitative. Unlike many ordinal preference languages, our language provides a facility to stipulate the relative strength of preferences. We claim that its qualitative nature facilitates elicitation.

Illustrative example: To help illustrate our preference language, consider the task of travel planning. A generic procedure, easily specified in Golog, might say: *In any order, book inter-city transportation, book local accommodations and book local transportation.* With this generic procedure in hand an individual user can specify their hard constraints (e.g., *Lara needs to be in Chicago July 29-Aug 5, 2009.*) together with a list of preferences described in the language to follow.

To understand the preference language, consider the composition we are trying to generate to be a situation – a sequence of actions or Web services executed from the initial situation. A user specifies his or her preferences in terms of a single, so-called *General Preference Formula.* This formula is an aggregation of preferences over constituent properties of situations (i.e., compositions). The basic building block of our preference formula is a *Trajectory Property Formula* which describes properties of (partial) situations (i.e., compositions).

Definition 1 (Trajectory Property Formula (TPF)).
A trajectory property formula is a sentence drawn from the smallest set $\mathcal{B}$ where:

1. $\mathcal{F} \subset \mathcal{B}$
2. $\mathcal{R} \subset \mathcal{B}$
3. $f \in \mathcal{F}$, *then* $\textbf{\textit{final}}(f) \in \mathcal{B}$
4. *If* $a \in \mathcal{A}$, *then* $\textbf{\textit{occ}}(a) \in \mathcal{B}$
5. *If* φ_1 *and* φ_2 *are in* $\mathcal{B}$, *then so are* $\neg\varphi_1$, $\varphi_1 \wedge \varphi_2$, $\varphi_1 \vee \varphi_2$, $(\exists x)\varphi_1$, $(\forall x)\varphi_1$, $\textbf{\textit{next}}(\varphi_1)$, $\textbf{\textit{always}}(\varphi_1)$, $\textbf{\textit{eventually}}(\varphi_1)$, *and* $\textbf{\textit{until}}(\varphi_1, \varphi_2)$.

$\mathbf{final}(f)$ states that fluent f holds in the final situation, $\mathbf{occ}(a)$ states that action a occurs in the present situation, and $\mathbf{next}(\varphi_1)$, $\mathbf{always}(\varphi_1)$, $\mathbf{eventually}(\varphi_1)$, and $\mathbf{until}(\varphi_1, \varphi_2)$ are basic LTL constructs.

TPFs establish properties of preferred situations (i.e., compositions of services). By combining TPFs using boolean connectives we are able to express a wide variety of properties of situations. E.g.[3]

$$\mathbf{final}(at(Home)) \tag{P1}$$

$$(\exists\, \boldsymbol{c}).\mathbf{occ}'(bookAir(\boldsymbol{c}, Economy, Direct)) \wedge member(\boldsymbol{c}, StarAlliance) \tag{P2}$$

$$\mathbf{always}(\neg((\exists\, \boldsymbol{h}).hotelBooked(\boldsymbol{h}) \wedge hilton(\boldsymbol{h}))) \tag{P3}$$

$$(\exists\, \boldsymbol{h}, \boldsymbol{r}).(\mathbf{occ}'(bookHotel(\boldsymbol{h}, \boldsymbol{r})) \wedge paymentOption(\boldsymbol{h}, Visa) \wedge starsGE(\boldsymbol{r}, 3) \tag{P4}$$

P1 states that the user is at home in the final situation. P2 states that at some point the user books a direct economy flight with a Star Alliance carrier. Recall there was no stipulation in the generic procedure regarding the mode of

[3] To simplify the examples many parameters have been suppressed. For legibility, variables are bold faced, we abbreviate $\mathbf{eventually}(\mathbf{occ}(\varphi))$ by $\mathbf{occ}'(\varphi)$, and we refer to the preference formulae by their labels.

transportation between cities or locally. P3 states that a Hilton hotel never be booked while P4 states that at some point the user books a hotel that accept Visa credit cards and has a rating of 3 or more.

To define a preference ordering over alternative properties of situations, we define *Atomic Preference Formulae* (APFs). Each alternative being ordered comprises two components: the property of the situation, specified by a TPF, and a *value* term which stipulates the relative strength of the preference.

Definition 2 (Atomic Preference Formula (APF)). *Let $\mathcal{V}$ be a totally ordered set with minimal element v_{min} and maximal element v_{max}. An atomic preference formula is a formula $\varphi_0[v_0] \gg \varphi_1[v_1] \gg ... \gg \varphi_n[v_n]$, where each φ_i is a TPF, each $v_i \in \mathcal{V}$, $v_i < v_j$ for $i < j$, and $v_0 = v_{min}$. When $n = 0$, atomic preference formulae correspond to TPFs.*

An APF expresses a preference over alternatives. Note that v_{min} is the most preferred and v_{max} is the least preferred. In what follows, we let $\mathcal{V} = [0, 1]$, but we could instead choose a strictly qualitative set like $\{best < good < indifferent < bad < worst\}$ since the operations on these values are limited to *max* and *min*. The following APFs express an ordering over Lara's preferences.

$$\begin{aligned}&\text{P2}[0]\\ &\quad \gg (\exists\, \boldsymbol{c}, \boldsymbol{w}).\mathbf{occ}'(\mathit{bookAir}(\boldsymbol{c}, \mathit{Economy}, \boldsymbol{w}) \wedge \mathit{member}(\boldsymbol{c}, \mathit{StarAlliance})[0.2]\\ &\qquad \gg \mathbf{occ}'(\mathit{bookAir}(\mathit{Delta}, \mathit{Economy}, \mathit{Direct}))[0.5] \end{aligned} \tag{P5}$$

$$\begin{aligned}&(\exists\, \boldsymbol{t}).\mathbf{occ}'(\mathit{bookCar}(\mathit{National}, \boldsymbol{t}))[0] \gg (\exists\, \boldsymbol{t}).\mathbf{occ}'(\mathit{bookCar}(\mathit{Alamo}, \boldsymbol{t}))[0.2]\\ &\quad \gg (\exists\, \boldsymbol{t}).\mathbf{occ}'(\mathit{bookCar}(\mathit{Avis}, \boldsymbol{t}))[0.8]\end{aligned} \tag{P6}$$

$$(\exists\, \boldsymbol{c}).\mathbf{occ}'(\mathit{bookCar}(\boldsymbol{c}, \mathit{SUV}))[0] \gg (\exists\, \boldsymbol{c}).\mathbf{occ}'(\mathit{bookCar}(\boldsymbol{c}, \mathit{Compact}))[0.2] \tag{P7}$$

P5 states that Lara prefers direct economy flights with a Star Alliance carrier, followed by economy flights with a Star Alliance carrier, followed by direct economy flights with Delta airlines. P6 and P7 are preference over cars. Lara strongly prefers National and then Alamo over Avis, followed by all other car-rental companies. Finally she slightly prefers an SUV over a compact with any other type of car a distant third.

To allow the user to specify more complex preferences and to aggregate preferences, General Preference Formulae (GFPs) extend our language to conditional, conjunctive, and disjunctive preferences.

Definition 3 (General Preference Formula (GPF)). *A formula Φ is a general preference formula if one of the following holds:*

- *Φ is an APF*
- *Φ is $\gamma : \Psi$, where γ is a TPF and Ψ is a GPF [Conditional]*
- *Φ is one of*
  - *$\Psi_0 \,\&\, \Psi_1 \,\&\, ... \,\&\, \Psi_n$ [General Conjunction]*
  - *$\Psi_0 \mid \Psi_1 \mid ... \mid \Psi_n$ [General Disjunction]*

where $n \geq 1$ and each Ψ_i is a GPF.

Continuing our example:

$$(\forall\, \boldsymbol{h}, \boldsymbol{c}, \boldsymbol{e}, \boldsymbol{w}).\mathbf{always}(\neg hotelBooked(\boldsymbol{h}) : \neg \mathbf{occ}'(bookAir(\boldsymbol{c}, \boldsymbol{e}, \boldsymbol{w}))) \quad \text{(P8)}$$

$$far \ : \ \text{P5} \quad \text{(P9)}$$

$$\text{P3}\,\&\,\text{P4}\,\&\,\text{P6}\,\&\,\text{P7}\,\&\,\text{P8}\,\&\,\text{P9} \quad \text{(P10)}$$

P8 states that Lara prefers not to book her air ticket until she has a hotel booked. P9 conditions Lara's airline preferences on her destination being far away. (If it is not far, she will not fly and the preferences are irrelevant.) Finally, P10 aggregates previous preferences into one formula.

Semantics: Informally, the semantics of our preference language is achieved through assigning a weight to a situation s with respect to a GPF, Φ, written $w_s(\Phi)$. This weight is a composition of its constituents. For TPFs, a situation s is assigned the value v_{min} if the TPF is satisfied in s, v_{max} otherwise. Recall that in our example above $v_{min} = 0$ and $v_{max} = 1$, though they could equally well have been a qualitative e.g., [excellent, abysmal]. Similarly, given an APF, and a situation s, s is assigned the weight of the best TPF that it satisfies within the defined APF. Returning to our example above, for P6 if a situation (composition) booked a car from Alamo rental car, it would get a weight of 0.2. Finally GPF semantics follow the natural semantics of boolean connectives. As such General Conjunction yields the maximum of its constituent GPF weights and General Disjunction yields the minimum of its constituent GPF weights. For a full explanation of the situation calculus semantics, please see [8]. Here we also define further aggregations that can be performed. These are mostly syntactic sugar that are compelling to the user and we omit them for space.

We conclude this section with the following definition which shows us how to compare two situations (and thus two compositions) with respect to a GPF:

Definition 4 (Preferred Situations). *A situation s_1 is at least as preferred as a situation s_2 with respect to a GPF Φ, written $pref(s_1, s_2, \Phi)$ if $w_{s_1}(\Phi) \leq w_{s_2}(\Phi)$.*

4 Web Service Composition

In this section, we define the notion of Web service composition with generic procedures and customizing user preferences, present an algorithm for computing these compositions and prove properties of our algorithm. Our definition relies on the definition of *Do* from (1) in Section 2.

Definition 5 (Web Service Composition w/ User Preferences (WSCP)). *A Web service composition problem with user preferences is described as a 5-tuple $(\mathcal{D}, O, \delta, C, \Phi)$ where:*

- *$\mathcal{D}$ is a situation calculus basic action theory describing functional properties of the Web services,*

- *O is a FOL theory describing the non-functional properties of the Web services*[4]*,*
- *δ is a generic procedure described in Golog,*
- *C is a formula expressing hard user constraints, and*
- *Φ is a GPF describing user preferences.*

A Web Service Composition (WSC) is a sequence of Web services $\boldsymbol{a}$ *such that*

$$\mathcal{D} \wedge O \models \exists s.Do(\delta, S_0, s) \wedge s = do(\boldsymbol{a}, S_0) \wedge C(s)$$

A preferred WSC (WSCP) is a sequence of Web services $\boldsymbol{a}$ *such that*

$$\begin{aligned}\mathcal{D} \wedge O \models \exists s.Do(\delta, S_0, s) \wedge s = do(\boldsymbol{a}, S_0) \wedge C(s)\\ \wedge \nexists s'.[Do(\delta, S_0, s') \wedge C(s') \wedge pref(s', s, \Phi)]\end{aligned}$$

A WSC is a sequence of Web services, $\boldsymbol{a}$, whose execution starting in the initial situation enforces the generic procedure and hard constraints terminating successfully in $do(\boldsymbol{a}, s)$. A WSCP yields a most preferred terminating situation.

4.1 Computing Preferred Compositions

A Golog program places constraints on the situation tree that evolves from S_0. As such, any implementation of Golog is effectively doing planning in a constrained search space, searching for a legal termination of the Golog program. The actions that define this terminating situation are the plan. In the case of composing web services, this plan is a web service composition.

To compute a preferred composition, WSCP, we search through this same constrained search space to find the *most preferred* terminating situation. Our approach, embodied in a system called GologPref, searches for this optimal terminating situation by modifying the PPLAN approach to planning with preferences proposed in [8]. In particular, GologPref performs best-first search through the constrained search space resulting from the Golog program, $\delta; C$. The search is guided by an admissible evaluation function that evaluates partial plans with respect to whether they satisfy the preference formula, Φ. The admissible evaluation function is the optimistic evaluation of the preference formula, with the pessimistic evaluation and the plan length used as tie breakers where necessary, in that order.

The preference formula Φ and the constraints C are evaluated over intermediate situations (partial compositions) by exploiting *progression* as described in [8]. Informally, progression takes a situation and a temporal logic formula (TLF), evaluates the TLF with respect to the state of the situation, and generates a new formula representing those aspects of the TLF that remain to be satisfied in subsequent situations.

Fig 1 provides a sketch of the basic GologPref algorithm following from PPLAN. The full GologPref algorithm takes as input a 5-tuple $(\mathcal{D}, O, \delta, C, \Phi)$. For ease of explication, our algorithm sketch in Fig 1 explictly identifies the initial situation of $\mathcal{D}$, *init*, the Golog program, $\delta; C$ which we refer to as *pgm* and Φ, which we refer to as *pref*. GologPref returns a sequence of Web services, i.e. a plan, and the weight of that plan. The *frontier* is a list of nodes of the form [*optW*, *pessW*, *pgm*,

[4] The content of $\mathcal{D}$ and O would typically come from an OWL-S, SWSO, or other semantic Web service ontology.

```
**GologPref**(*init*, *pgm*, *pref*)
*frontier* ← **initFrontier**(*init*, *pgm*, *pref*)
**while** *frontier* $\neq \emptyset$
    *current* ← **removeFirst**(*frontier*)
    % establishes current values for *progPgm*, *partialPlan*, *state*, *progPref*
    **if** *progPgm*=*nil* and *optW*=*pessW*
        **return** *partialPlan*, *optW*
    **end if**
    *neighbours* ← **expand**(*progPgm*, *partialPlan*, *state*, *progPref*)
    *frontier* ← **sortNmergeByVal**(*neighbours*, *frontier*)
**end while**
**return** [], $\infty$
```

expand(*progPgm*, *partialPlan*, *state*, *progPref*) returns a list of new nodes to add to the frontier. If *partialPlan*=*nil* then **expand** returns [ ]. Otherwise, **expand** uses Golog's *Trans* to determine all the executable actions that are legal transitions of *progPgm* in *state* and to compute the remaining program for each.
It returns a list which contains, for each of these executable actions a a node
(*optW*, *pessW*,*newProgPgm*, *newPartialPlan*, *newState*, *newProgPref*)
and for each a leading to a terminating state, a second node
(*realW*, *realW*, *nil*, *newPartialPlan*, *newState*, *newProgPref*).

Fig. 1. A sketch of the GologPref algorithm

partialPlan, *state*, *pref*], sorted by optimistic weight, pessimistic weight, and then by length. The frontier is initialized to the input program and the empty partial plan, its *optW*, *pessW*, and *pref* corresponding to the progression and evaluation of the input preference formula in the initial state.

On each iteration of the **while** loop, GologPref removes the first node from the frontier and places it in *current*. If the Golog program of *current* is *nil* then the situation associated with this node is a terminating situation. If it is also the case that *optW*=*pessW*, then GologPref returns *current*'s partial plan and weight. Otherwise, it calls the function **expand** with *current's* node as input.

expand returns a new list of nodes to add to the frontier. If *progPgm* is *nil* then no new nodes are added to the frontier. Otherwise, **expand** generates a new set of nodes of the form [*optW*, *pessW*, *prog*, *partialPlan*, *state*, *pref*], one for each action that is a legal Golog transition of *pgm* in *state*. For actions leading to terminating states, **expand** also generates a second node of the same form but with *optW* and *pessW* replaced by the actual weight achieved by the plan. The new nodes generated by **expand** are then sorted by *optW*, *pessW*, then length and merged with the remainder of the frontier. If we reach the empty frontier, we exit the **while** loop and return the empty plan.

We now prove the correctness of our algorithm.

Theorem 1 (Soundness and Optimality).
Let $\mathcal{P}$=$(\mathcal{D}, O, \delta, C, \Phi)$ be a Web service composition problem, where δ is a tree program. Let $\boldsymbol{a}$ *be the plan returned by GologPref from input $\mathcal{P}$. Then* $\boldsymbol{a}$ *is a WSCP of $(\mathcal{D}, O, \delta, C, \Phi)$.*

Proof sketch: We prove that the algorithm terminates appealing to the fact that δ is a tree program. Then we prove that $\boldsymbol{a}$ is a WSC by cases over *Trans* and *Final*. Finally we prove that $\boldsymbol{a}$ is also optimal, by exploiting the correctness of progression of preference formuale proven in [8], the admissibility of our evaluation function, and the bounded size of the search space generated by the Golog program $\delta; C$.

4.2 Integrated Optimal Web Service Selection

Most Web service composition systems use AI planning techniques and as such generally ignore the important problem of Web service selection or discovery, assuming it will be done by a separate matchmaker. The work presented here is significant because it enables the selection of services for composition based, not only on their inputs, outputs, preconditions and effects but also based on other nonfunctional properties. As such, users are able to specify properties of services that they desire along side other properties of their preferred solution, and services are selected that optimize for the users preferences in the context of the overall composition.

To see how selection of services can be encoded in our system, we reintroduce the service parameter $\boldsymbol{u}$ which was suppressed from the example preferences in Section 3. Revisiting P2, we see how the selection of a service $\boldsymbol{u}$ is easily realized within our preference framework with preference P2'.

$$(\exists\, \boldsymbol{c}, \boldsymbol{u}).\mathbf{occ}'(\mathit{bookAir}(\boldsymbol{c}, \mathit{Economy}, \mathit{Direct}, \boldsymbol{u})) \wedge \mathit{member}(\boldsymbol{c}, \mathit{StarAlliance}) \\ \wedge\ \mathit{serviceType}(\boldsymbol{u}, \mathit{AirTicketVendor}) \wedge \mathit{sellsTickets}(\boldsymbol{u}, \boldsymbol{c}) \qquad \text{(P2')}$$

P2' causes GologPref to prefer booking air tickets with an air ticket vendor that sells the tickets of a carrier that is a member of Star Alliance.

5 Implementation and Application

We have implemented the generation of Web Service compositions using generic procedures and customizing user preferences as described in previous sections. Our implementation, GologPref, builds on an implementation of PPLAN[8] and an implementation of IndiGolog [6] both in SWI Prolog[5].

GologPref interfaces with Web services through the implementation of domain-specific scrapers developed using AgentBuilder 3.2, and AgentRunner 3.2, Web agent design applications developed by Fetch Technologies ©. Among the sites we have scraped are Mapquest, and several air, car and hotel services. The information gathered is collected in XML and then processed by GologPref.

We tested GologPref in the domain of travel planning. Our tests serve predominantly as a proof of the concept and to illustrate the utility of GologPref.

Our generic procedure which is represented in Golog was very simple, allowing flexibility in how it could be instantiated. What follows is an example of the Prolog encoding of a GologPref generic procedure.

[5] See [6] for a description of the translation of $\mathcal{D}$ to Prolog.

```
anyorder[bookAcc, bookCityToCityTranspo, bookLocalTranspo]

proc(bookAcc(Location, Day, Num),
[ stayWithFriends(Location) | bookHotel(Location, Day, Num) ]).

proc(bookLocalTranspo(Location, StartDay, ReturnDay),
[       getRide(Location, StartDay, ReturnDay)  |
        walk(Location)   |   bookCar(Location, StartDay, ReturnDay)  ]).

proc(bookCityToCityTranspo(Location, Des, StartDay, ReturnDay),
[       getRide(Location, Des, StartDay, ReturnDay) |
        bookAir(Location, Des, StartDay, ReturnDay) |
        bookCar(Location, Des, StartDay, ReturnDay) ]).
```

We tested our GologPref generic procedure with 3 different user profiles: Jack the impoverished university student, Lara the picky frequent flyer, and Conrad the corporate executive who likes timely luxury travel. Each user lived in Toronto and wanted to be in Chicago for specific days. A set of rich user preferences were defined for each user along the lines of those illustrated in Section 3. These preferences often required access to different Web information, such as driving distances.

Not surprisingly, in all cases, GologPref found the optimal WSC for the user. Compositions varied greatly ranging from Jack who arranged accommodations with friends; checked out the distance to his local destinations and then arranged his local transportation (walking since his local destination was close to where he was staying); then once his accommodations were confirmed, booking an economy air ticket Toronto-Chicago with one stop on US Airways with Expedia. Lara on the other hand, booked a hotel (not Hilton), booked an intermediate-sized car with National, and a direct economy air ticket with Star Alliance partner Air Canada via the Air Canada Web site. The optimality and the diversity of the compositions, all from the same generic procedure, illustrate the flexibility afforded by the WSCP approach.

Figure 2 shows the number of nodes expanded relative to the search space size for 6 test scenarios. The full search space represents all possible combinations of city-to-city transportation, accommodations and local transportation available to the users which could have been considered. These results illustrate the effectiveness of the heuristic used to find optimal compositions.

Case Number	**Nodes Expanded**	**Nodes Considered**	**Time (sec)**	**Nodes in Full Search Space**
1	104	1700	14.38	28,512
2	102	1647	13.71	28,512
3	27	371	2.06	28,512
4	27	368	2.09	28,512
5	99	1692	14.92	28,512
6	108	1761	14.97	28,512

Fig. 2. Test results for 6 scenarios run under 64bit Ubuntu Linux with 2.66 GHz CPU

6 Summary and Related Work

In this paper we argued that the integration of user preferences into Web service composition was a key missing component of Web service composition. Building on our previous framework for Web service composition via generic procedures [2] and our work on preference-based planning [8], we proposed a system for Web service composition with user preferences. Key contributions of this paper include: characterization of the task of Web service composition with generic procedures and user preferences, provision of a previously developed language for specifying user preferences, provision of the GologPref algorithm that integrates preference-based reasoning into Golog, a proof of the soundness and optimality of GologPref with respect to the user's preferences, and a working implementation of our GologPref algorithm. A notable side effect of our framework is the seamless integration of Web service selection with the composition process.

We tested GologPref on 6 diverse scenarios applied to the same generic procedure. Results illustrated the diversity of compositions that could be generated from the same generic procedure. The number of nodes expanded by the heuristic search was several orders of magnitude smaller than the grounded search space, illustrating the effectiveness of the heuristic and the Golog program in guiding search.

A number of researchers have advocated using AI planning techniques to address the task of Web service composition including using regression-based planners [3], planners based on model checking (e.g., [4]), highly optimized hierarchical task network (HTN) planners such as SHOP2 (e.g., [18]), and a combination of classical and HTN planning called XPLAN [14]. Like Golog, HTNs afford the user the ability to define a generic procedure or *template* of how to perform a task.

Sirin et al. incorporated simple service preferences into the SHOP2 HTN planner to achieve dynamic service binding [7]. Their preference language is significantly less expressive than the one presented here and is restricted to the task of service selection rather than solution optimization. Nevertheless, it is a promising start. Also related is the work by Fritz and the third author in which they *precompiled* a subset of the preference language presented here into Golog programs that were then integrated with a decision-theoretic Golog (DTGolog) program [19]. The main objective of this work was to provide a means of integrating qualitative and quantitative preferences for agent programming. While both used a form of Golog, the form and processing of preferences was quite different.

Since the original publication of this work, preference-based planning has been the subject of much interest, spurred on in great part by three tracks on planning with preferences at the 2006 International Planning Competition (IPC-5). A number of preference-based planners were developed, including one by a subset of the authors, all based on the competition's PDDL3 language [20]. The most notable new work that is directly related to this paper is that of [21]. In this paper, the authors propose a prototype HTN preference-based planner, **scup**, tailored to the task of Web service composition and that uses PDDL3 as its preference specification language.

We also have two follow-up pieces of work [22] and [23] in which we specify flexible templates in the form of an HTN rather than a Golog generic procedure. In [22] we proposed a qualitative language very similar to the preference language discussed in this paper but tailored to HTN planning. In [23] we extended PDDL3 with HTN-specific preference constructs. The proposed planners employ state of the art heuristic guided search and algorithms that exploit HTN-specific preferences and control. In contrast to the work presented here, optimality is not guaranteed without exhaustive search. In future work, we would like to improve the GologPref algorithm with the addition of more informative inadmissible heuristics coupled with branch and bound search. We would also like to exploit a recent extension to the $\mathcal{LPP}$ preference language to include preferences over the occurrence of Golog complex actions [9].

Acknowledgements

The authors would like to thank Meghyn Bienvenu for her initial work on PPLAN which was fundamental to the realization of this work. We also wish to thank Christian Fritz for subsequent improvements to the PPLAN algorithm and for much useful discussion. Finally we would like to thank Fetch Technologies for allowing us to use their AgentBuilder software. We gratefully acknowledge funding from the Natural Sciences and Engineering Research Council of Canada (NSERC), the Ontario Ministry of Innovations Early Researcher Award (ERA), and the CRA's Canadian Distributed Mentorship Project (CDMP).

References

1. Sohrabi, S., Prokoshyna, N., McIlraith, S.A.: Web service composition via generic procedures and customizing user preferences. In: Cruz, I., Decker, S., Allemang, D., Preist, C., Schwabe, D., Mika, P., Uschold, M., Aroyo, L.M. (eds.) ISWC 2006. LNCS, vol. 4273, pp. 597–611. Springer, Heidelberg (2006)
2. McIlraith, S., Son, T.: Adapting golog for composition of semantic web services. In: Proceedings of the 8th International Conference on Knowledge Representation and Reasoning (KR), Toulouse, France, pp. 482–493 (2002)
3. McDermott, D.V.: Estimated-regression planning for interactions with web services. In: Proceedings of the 6th International Conference on Artificial Intelligence Planning and Scheduling (AIPS), pp. 204–211 (2002)
4. Traverso, P., Pistore, M.: Automated composition of semantic web services into executable processes. In: McIlraith, S.A., Plexousakis, D., van Harmelen, F. (eds.) ISWC 2004. LNCS, vol. 3298, pp. 380–394. Springer, Heidelberg (2004)
5. McIlraith, S., Son, T., Zeng, H.: Semantic Web services. IEEE Intelligent Systems (Special Issue on the Semantic Web) 16 (2001)
6. Reiter, R.: Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems. MIT Press, Cambridge (2001)
7. Sirin, E., Parsia, B., Wu, D., Hendler, J., Nau, D.: HTN planning for web service composition using SHOP2. Journal of Web Semantics 1(4), 377–396 (2005)
8. Bienvenu, M., Fritz, C., McIlraith, S.: Planning with qualitative temporal preferences. In: Proceedings of the 10th International Conference on Knowledge Representation and Reasoning (KR), pp. 134–144 (2006)

9. Bienvenu, M., Fritz, C., McIlraith, S.: Specifying and generating preferred plans (submitted for publication, 2009)
10. Horrocks, I., Patel-Schneider, P., van Harmelen, F.: From $\mathcal{SHIQ}$ and RDF to OWL: The making of a web ontology language. Journal of Web Semantics 1(1), 7–26 (2003)
11. Martin, D., Burstein, M., McDermott, D., McIlraith, S., Paolucci, M., Sycara, K., McGuinness, D., Sirin, E., Srinivasan, N.: Bringing semantics to web services with OWL-S. World Wide Web Journal 10(3), 243–277 (2007)
12. Bruijn, J.D., Lausen, H., Polleres, A., Fensel, D.: The web service modeling language WSML: An overview. Technical report, DERI (2006)
13. Battle, S., Bernstein, A., Boley, H., Grosof, B., Gruninger, M., Hull, R., Kifer, M., Martin, D., McIlraith, S., McGuinness, D., Su, J., Tabet, S.: Semantic web service ontology (SWSO) first-order logic ontology for web services, FLOWS (2005), http://www.daml.org/services/swsl/report/
14. Klusch, M., Gerber, A., Schmidt, M.: Semantic web service composition planning with OWLS-Xplan. In: AAAI 2005 Fall Symposium (2005)
15. McIlraith, S., Fadel, R.: Planning with complex actions. In: Proceedings of the 9th International Workshop on Non-Monotonic Reasoning NMR-2002, pp. 356–364 (2002)
16. Narayanan, S., McIlraith, S.: Simulation, verification and automated composition of web services. In: Proc. of the 11th International World Wide Web Conference, WWW 2002 (2002)
17. De Giacomo, G., Lespérance, Y., Levesque, H.: ConGolog, a concurrent programming language based on the situation calculus. Artificial Intelligence 121(1–2), 109–169 (2000)
18. Sirin, E., Parsia, B., Hendler, J.: Template-based composition of semantic web services. In: AAAI-2005 Fall Symposium on Agents and the Semantic Web (2005)
19. Fritz, C., McIlraith, S.: Decision-theoretic golog with qualitative preferences. In: Proceedings of the 10th International Conference on Knowledge Representation and Reasoning (KR), Lake District, UK, pp. 153–163 (2006)
20. Gerevini, A., Long, D.: Plan constraints and preferences for PDDL3. Technical Report 2005-08-07, Department of Electronics for Automation, University of Brescia, Brescia, Italy (2005)
21. Lin, N., Kuter, U., Sirin, E.: Web service composition with user preferences. In: Bechhofer, S., Hauswirth, M., Hoffmann, J., Koubarakis, M. (eds.) ESWC 2008. LNCS, vol. 5021, pp. 629–643. Springer, Heidelberg (2008)
22. Sohrabi, S., McIlraith, S.A.: On planning with preferences in HTN. In: 12th International Workshop on Non-Monotonic Reasoning (NMR-2008), Sydney, Australia, pp. 241–248 (2008)
23. Sohrabi, S., Baier, J., McIlraith, S.: HTN planning with preferences. In: Proceedings of the 21st International Joint Conference on Artificial Intelligence, IJCAI (2009)

Dealing with Complexity Using Conceptual Models Based on *Tropos*

Jaelson Castro[1], Manuel Kolp[2], Lin Liu[3], and Anna Perini[4]

[1] Universidade Federal de Pernambuco, Recife, Brazil
jbc@cin.ufpe.br
[2] University of Louvain, LSM-ISYS, Louvain-la-Neuve, Belgium
manuel.kolp@uclouvain.be
[3] Tsinghua University, Beijing, China
linliu@tsinghua.edu.cn
[4] Fondazione Bruno Kessler – Irst, CIT, Trento, Italy
perini@fbk.eu

Abstract. Since its establishment, the major objective of the Tropos methodology has been to develop an approach for the systematic engineering of agent-oriented information systems. In this chapter we illustrate a number of approaches to deal with complexity, which address different activities in software development and are deemed to be used in combination. We begin with handling complexity at requirements levels. In particular we examine how modularization can be improved using some of Aspect Oriented Software Development Principles. We then examine how model-based testing applied in parallel to requirements analysis and design can support incremental validation and testing of software components, as well as help to clarify ambiguities in requirements. We also look at how Tropos can help to address complexity in social context when making design decisions. Last but not least, we show how to tackle complexity at the process modelling level. We explore iterative development extension to Tropos as well as perspectives taken from software project management. This allows us to deal with the complexity of large real world projects.

Keywords: requirements engineering, complexity, aspect, goal modelling, testing, process modelling.

1 Introduction

Enterprises are continually changing their internal structures and business processes, as well as their external alliances, as they strive to improve and grow. Software systems that operate within such a setting have to evolve continuously to accommodate new technologies and meet new requirements. Indeed, it is well known that the latest generation of software systems, such as Enterprise Resource Planning (ERP), groupware, knowledge management and e-business systems, should be designed to perform within ever-changing organizational environments.

These features will characterize more and more future software systems [14], which will be employed by an increasing number of people for different purposes and using a variety of devices. These systems will use a growing amount of data stored, accessed, manipulated, and refined in a distributed way, and rest on a set of interdependencies among software components (services), which may dynamically change. This increase in size (scale) and dynamicity are considered major sources of complexity of software systems, which calls for new solution approaches and new concepts for system design, development, operation, and evolution. A rich research agenda is proposed in [14] that highlights, for example, the need of more expressive modeling languages and of revisiting model-based, aspect-oriented, and other generative methods for helping to validate and certify requirements.

The Tropos project [24], [5] was launched with the objective to develop a methodology for building agent-oriented information systems, in competition with existing methodologies founded on structured and object-oriented concepts. Agent-based technologies are considered a promising solution towards the realization of software having flexibility and (self-)adaptive properties [23], moreover the agent paradigm offers a suitable set of concepts to model large-scale systems in terms of socio-technical ecosystems [14].

The Tropos methodology rests on the idea of starting by building a model of the organizational context within which the system-to-be will eventually function, then the system-to-be is introduced and the model is incrementally refined and extended with a definition of the functional and non-functional requirements of the system-to-be. This model provides a common interface to the various software development activities. The model also serves as a basis for documentation and evolution of the software system. The approach is requirements-driven in the sense that the concepts used to define requirements for a software system are also used later on during design and implementation. To this end, Tropos adopts the concepts offered by *i** [37], a modelling framework proposing concepts such as *actor* (actors can be *agents*, *positions* or *roles*), social dependency among actors, including *goal*, *softgoal*, *task* and *resource* dependencies. Thus, an actor can depend upon another one to satisfy a goal, execute a task, and provide a resource or satisfice a softgoal. Softgoals are associated to non-functional requirements, while goals, tasks and resources are associated to system functionalities. The i* framework offers two models: the Strategic Dependency (SD) and Strategic Rationale (SR). The SD model consists of a set of nodes and links connecting them, where nodes represent actors and each link indicates a dependency between two actors (dependum). The depending actor is called depender, and the actor who is depended upon is called the dependee. The SR model provides a more detailed level of modeling by looking "inside" actors to model internal intentional relationships.

Researches on how to manage software complexity with Tropos have been conducted in parallel, by different research groups. They attack the problem from a number of perspectives, involving technical (e.g. related to issues in modeling and validation of requirements), as well as managerial and automation issues.

In this chapter, we illustrate some of them, with the objective to show their complementarities and the potential to be used in combination.

It is well known that requirements models may become cluttered, compromising their evolution and scalability. In fact, empirical evaluation has shown that there is a lack of modularity in the *i** framework [13]. This is a serious drawback for large and complex projects. In fact, i* models tend to include scattered and tangled representations, i.e. crosscutting, resulting in models with poor modularization and, therefore, harder to understand and maintain [1]. In Section 3, we present an approach to create modular *i*/Tropos* models, where a desired concern is separated as an individual actor. Consequently, improved modularization mechanisms are required to avoid the crosscutting representations in *i* /Tropos* models.

Developing complex software systems requires that single components, as well as the overall system, are incrementally validated and certified against requirements and user expectations, along the whole development process. This motivated the adoption of a V-model[1] approach to software development in Tropos that complement analysis and design with validation phases, which is called Goal-Oriented Software Testing (GOST) [26]. Section 4 recalls basic elements of GOST and illustrates how it works when validating early requirements against user goals. A further benefit of the GOST methodology will also emerge since early test specification will require clarifying ambiguities in the requirements model, then improving the requirements model itself.

In order to design a better information system, a designer would like to have notations to visualize how design experts' know-how can be applied according to one's specific social and technology situation. Section 5 proposes the combined use of *Tropos* and a scenario-oriented notation *UCM* for representing design knowledge of information systems. So that goal models are combined with scenarios descriptions to complement each other to handle complexity in design decision making. The combined use of GRL (a variant of *i**/Tropos) and a scenario mapping approach is part of the Users Requirements Notation (URN), a newly approved ITU-T standard [17] [1][22].

Section 6 proposes *I-Tropos*, a software project management framework dedicated to extend *Tropos* with an iterative life cycle. The process fills the project and product life cycle gaps of Tropos and offers a goal-oriented project management perspective to support project stakeholders for applying Tropos on large information systems. It is supported by *DesCARTES* a specific CASE tool.

2 Running Example

In order to illustrate Tropos requirements models, let us consider the Media Shop example presented in [10]. *Media Shop* is a store which sells and ships different kinds of media items, such as books, newspapers, magazines. To increase market share, *Media Shop* has decided to use the *Medi@* system, a business to customer retail sales front-end on the Internet to allow an on-line customer to examine the items in its catalogue and place orders.

The system uses communication facilities provided by *Telecom Cpy*. There are no registration restrictions, or identification procedures for Medi@ users. Potential customers can search the on-line store by either browsing the catalogue or querying the item database. An on-line search engine allows customers with particular items in mind to search title, author/artist and description fields through keywords or full-text

[1] The V-Model gets its name from the fact that the process is often mapped out as a flowchart that takes the form of the letter V: the left edge defines a sequence of analysis and design activities, the right edge the corresponding set of validation and testing activities.

search. If the item is not available in the catalogue, the customer has the option of asking *Media Shop* to order it.

In Figure 1 you can find an expanded description of the *Medi@ actor*. In this actor, a root task *Manage Internet Shop* is specified (located at the centre-top of the larger circle that represents the *Medi@* actor's boundary). That task is firstly refined into *Item Searching Handled* goal, *Secure, Adaptable* and *Available* softgoals, and *Produce Statistics* task. These intentional elements are further refined by using task-decomposition, means-end and contribution links to define the *Medi@* system requirements. These three new types of relationships are explained as follows: (i) task-decomposition links describe what should be done to perform a certain task (e.g., the relationship between the *Provide Access Link* task and the *Provided Internet Service* task inside the *Telecom Cpy* actor); (ii) means-end links suggest that one model element can be offered as a means to achieve another model element (e.g., relationship between the *Chose Non-Available Item* task and the *Item Selection* goal inside the Medi@ actor); (iii) contributions links suggest how a task can contribute (positively or negatively) to satisfy a softgoal (e.g., the relationship between the *Use Fault-Tolerant Strategies* task and the *Available* softgoal inside the Medi@ actor).

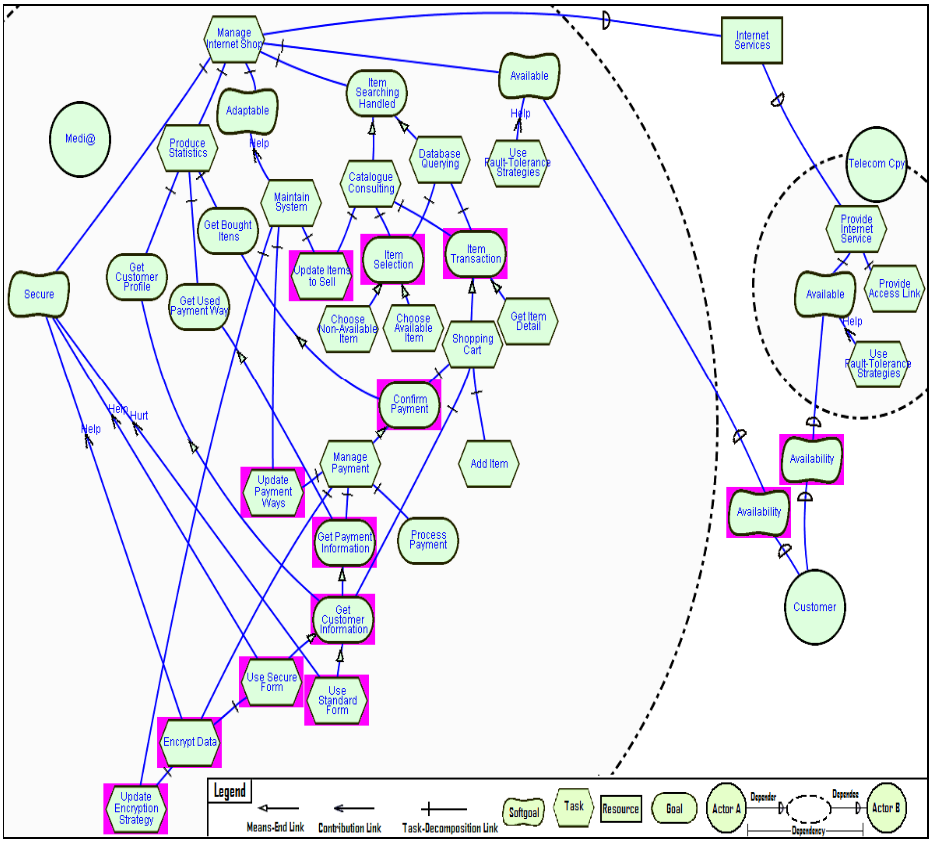


Fig. 1. The Medi@ Strategic Rationale Model

Apart from the previous three types of relationships, there are intentional dependencies between actors, which can be of four types: goal, task, resource or softgoal. For example, the *Customer* actor (depender) is related to *Medi@* (dependee) actor through *Availability* goal (dependum).

3 Modularization of Requirements Models

As the problem at hand grows, *i*/Tropos* models may become cluttered, compromising their evolution and scalability. This is a serious drawback for large and complex projects. In fact, *i** models tend to include scattered and tangled representations, i.e. crosscutting, resulting in models with poor modularization and, therefore, harder to understand and maintain [1]. This problem could be avoided if some approach was available to create modular *i*/Tropos* models, where a desired concern is separated as an individual actor. Consequently, improved modularization mechanisms are required to avoid the crosscutting representations in i*/Tropos models.

In the sequel we introduce (i) a set of guidelines to identify crosscutting concerns in i* models; and (ii) propose an extension of the *i** modelling language [37] by adding aspectual constructors to modularize crosscutting concerns and to allow its graphical composition with other system modules.

3.1 Identifying and Modularizing Aspects

The following three guidelines described helps to identify aspectual elements.

Guideline G1 (Repeated dependum): *if a dependum (i.e. a goal, a task, a resource or a softgoal) is provided by at least two dependee actors, and the subgraph operationalizing that dependum is handled equally by all dependee actors, then this operationalization is part of an aspectual element.*

This guideline aims at identifying dependencies in a SD model that have been repeated and addressed in similar ways. Thus, if a dependency has multiple similar occurrences, i.e. different dependee actors can handle (operationalize) it in the same way, the elements contained in the operationalization sub-graph can be relocated to an aspectual element. For example, in Figure 1, the *Availability* softgoal dependum has multiple occurrences in the model. But this repetition is not sufficient. It is also necessary to check if their respective operationalizations (inside the respective dependee actors) are the same. Notice that in *Telecom Cpy a*ctor, the *Availability* dependum is related to the softgoal *Available*, operationalized by *Use Fault-Tolerance Strategies* task. The *Medi@* actor treats *Availability* dependum similarly, since it is also operationalized by *Use Fault-Tolerance Strategies* task. This means that the entire sub-graph operationalizing the *Availability* dependum will be part of an aspectual element, because it is repeated in different actors. This element will be the *Availability Manager* aspectual element (Figure 3).

Guideline G2 (Repeated intentional element): This guideline is subdivided into three sub-guidelines: one deals with the task decomposition link; another deals with means-ends links (which includes the contribution link); and the last one deals with both links (task-decomposition and means-ends links). These sub-guidelines are applied to the intentional elements that are internal to the actor's boundary presented in the SR model.

Guideline G2.1 (Intentional element in a task-decomposition link): *if an intentional element (goal, softgoal or task) is required by (i.e., is a decomposition element of) two or more internal tasks, indicating a sharing of information, then the subgraph that contains this element as the root is part of an aspectual element.*

In Figure 1, the *Item Selection* goal is simultaneously required through task-decomposition by the tasks: *Database Querying* and *Catalogue Consulting*. Thus, the *Item Selection* goal is part of an aspectual element.

Guideline G2.2 (Intentional element in a means-end link): *if an intentional element (goal, softgoal or task) is a means element which is required by two or more end elements (indicating a sharing of information) then the sub-graph containing this element as a root will be part of an aspectual element.*

According to Figure 1, *Use Secure Form* task is simultaneously a means to *Get Customer Information* goal (end) and contributes to *Secure* softgoal. Hence, if we consider a contribution link as a means-end link (such as in [37]), the *Use Secure Form* is part of an aspectual element.

Guideline G2.3 (Intentional element is found simultaneously in a task-decomposition link and in a means-end link): *if an intentional element (goal, softgoal or task) is a means element and also is a sub-element in a task-decomposition (indicating a sharing of information) then the sub-graph containing this element as the root is part of an aspectual element.*

In Figure 1, this guideline captures the *Get Payment Information* goal which is a means to achieve *Get Used Payment Way* goal (through a means-end link) and a sub-element of *Manage Payment* task (through a task-decomposition link). Then the sub-graph with *Get Payment Information* goal as the root is part of an aspectual element.

Guideline G3 (Redundancy): *the aspectual elements identified by guidelines G1 and G2 are now merged together to remove multiple occurrences.*

In the example, we have captured *Encrypt Data* task simultaneously by the guidelines G2.2 and G2.3. To increase cohesion of the aspectual element, along with each intentional element identified by the guidelines, it is also required to extract other intentional elements related to the same concern and locate them all into the same aspectual element.

For example, in Figure 1, the tasks *Update Encryption Strategy* and *Encrypt Data* were identified by the guidelines and should be modularized by an aspectual element. However, these tasks are part of the sub-graph that operationalizes the *Secure* softgoal which was first identified, separated and located into an aspectual element. Therefore, those tasks can be seen as related to the *Security* concern. Thus, all these intentional elements can be extracted and located into the aspectual element *Security Manager* (Figure 2 (b)).

3.2 Identifying Relationship among Aspectual Elements

In parallel with the aspect identification, we may store the information of the relationships of all elements captured by the guidelines (for example into a table) to allow the automation of the process From the model in Figure 1 and the guidelines proposed in Section 3.1 we list in Table 1 some elements (for lack of space, not all captured elements are listed in the Table 1) that must be made persistent: (i) the dependee actor, which provides an intentional element, in both SD and SR models; (ii) the aspect which is the identified crosscutting intentional element; (iii) the concern addressed by intentional element; (iv) the related elements which represent the elements of the subgraph provided that they have not already been captured as aspect; (v) the chosen name for the identified aspectual element.

For example, in the case study the *Customer* depends on the *Medi@* and *Telecom Cpy* for the intentional *Availability* element (a softgoal). According to guideline G1 the characteristics of this dependency indicated the need to identify an aspect to deal with the availability concern. Hence, we need to define an appropriate name for it. For example it could be called *Availability Manager*. Its related intentional element is only the *Use Fault-Tolerance Strategies* task that is present in both original dependee actors (*Medi@* and *Telecom Cpy*). Hence, it should be transferred to that new element (*Availability Manager*). Similar analysis can be performed for the *Confirm Payment* goal (by G2.3), *Encrypt Data* task (by G2.3), and *Item Selection* task (by G2.1).

Table 1. Modularization of Aspects

Actor	Crosscutting Element	Concern	Related Elements	Aspectual Element Name
Medi@	*Confirm Payment* goal	*Payment*	*Manage Payment* task, and *Process Payment* and *Get Payment Information* goals	*Payment Processor*
Medi@	*Encrypt Data* task	Security	*Update Encryption Strategy task,* Secure softgoal	*Security Manager*
Medi@	*Item Selection* goal	*Item Selection*	*Choose Available Item* and *Choose Non-Available Item* tasks	*Item Selector*
Medi@, Telecom Cpy	*Available softgoal*	*Availability*	*Use Fault-Tolerance Strategies*	*Avalability Manager*

3.3 Representing Aspectual Elements Using the Aspectual i* Notation

A specific notation has been created to represent aspectual i* models. This leads to the addition of two new concepts in the *i** modeling language, namely aspectual element and crosscut relationship. Aspectual elements modularize crosscutting concerns and the crosscut relationship captures the information of source and target model elements, as well as, when and how an aspectual element crosscuts other model elements. For modularization purposes and following the principles of AOSD, we should extract and modularize the aspects, removing them from the original actors, and placing them in a new type of model element, the so called *Aspectual Element*. This new element is graphically represented by an actor with a vertical line crossing it (see, for example, the Security Manager element in Figure 2). An aspectual element, as well as an actor, is composed of intentional elements, whereas an intentional element can be a goal, a softgoal, a resource or a task. An aspectual element can be composed with an actor or another aspectual element through a *Crosscut Relationship*. This relationship specifies how an intentional element, located inside an aspectual element, is related with another intentional element, which is located inside an actor or another aspectual

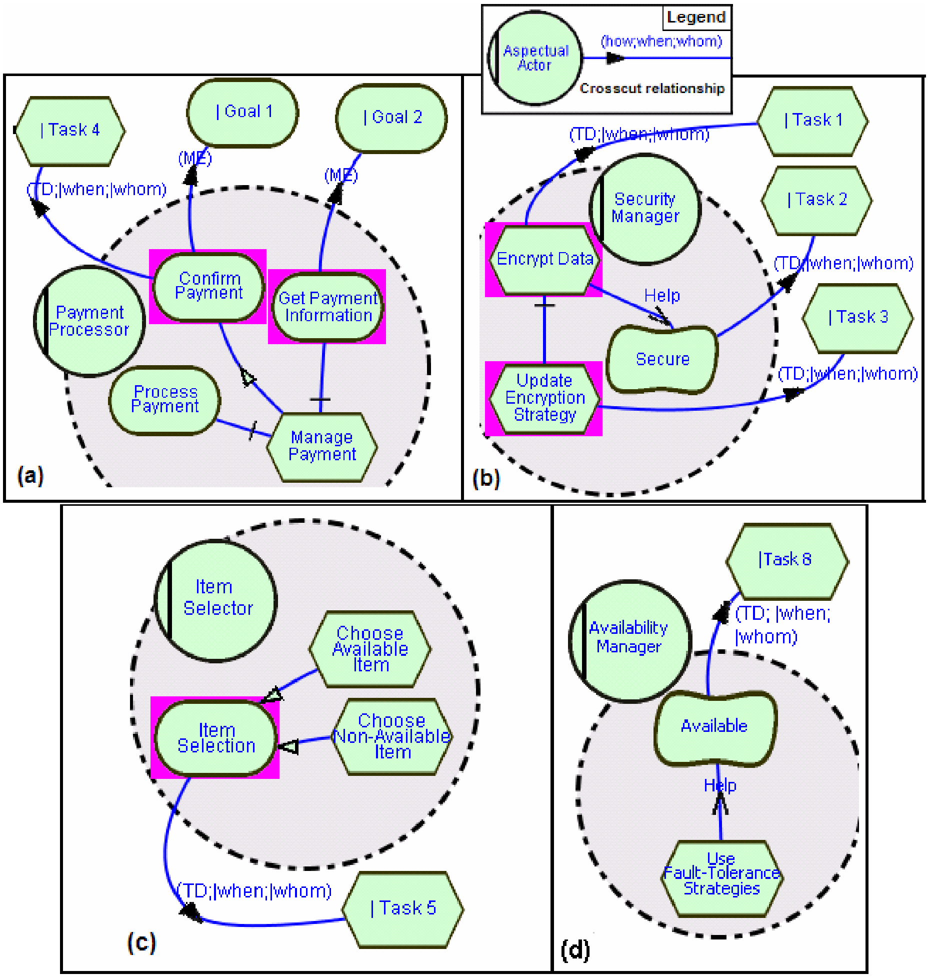


Fig. 2. Aspect Modelling for: (a) Payment Processor; (b)Security Manager; (c) Item Selector; (d) Availability Manager

element. The *how* attribute present in the crosscut relationship means the type of i* relationship (*Task-Decomposition (TD)*, *Means-End (ME)* and *Contribution*) that will be recovered with the weaving. The crosscut relationship between each aspectual element and other model elements are shown as arcs, with a dark triangle (Figure 2).

The direction indicated by the triangle suggests the way of the composition, meaning that the source element's behavior needs to be composed with the target elements' behaviors. The crosscut relationship also contains a when attribute, which can assume the values before or after, to specify when an element inside the aspectual element will be composed with an element inside another aspectual element or Actor. This composition rule must be defined taking into account the intentional element in relation to whom the composition must occur, described by the attribute whom (see, for example, the legend in Figure 2).

In order to describe the aspectual elements and to systematically compose them with other model elements, we use the concept of model roles [19] which have been used to describe object-oriented patterns, as proposed in [15], and agent-oriented patterns, as presented in [32]. They facilitate the graphical composition of concern and improve the reuse of aspectual elements.

In particular, to describe aspectual elements, it is necessary to specialize each target intentional element in a crosscut relationship and the attribute of the crosscut relationship: how, when and whom. Model roles are identified by preceding the intentional elements (goals, task, softgoals) identifiers with a "|" (see Figure 2). In practice they work as variables to be instantiated to concrete model elements.

Let us concentrate on *Payment Processor, Security Manager*, *Item Selector*, and *Availability Manager* depicted in Figure 2(a), Figure 2(b), Figure 2(c) and Figure 2(d), respectively. The composition of the aspectual elements with the original model requires the instantiation (or binding) of the model roles present in the crosscut relationship and in the target element related with the crosscut relationship. Thus, to compose the aspectual element *Payment Processor* with the *Medi@* actor, we need to bind |Goal 2 to *Get Used Payment Way* (see Figure 3). Since the relationship from a goal to another goal can only be a means-end link, the properties of that crosscut relationship do not have any model roles. Observe that the *when* and *whom* properties are used just in case we need to insert the ordering of composition. If the *when* (and, therefore, the *whom*) property is empty, then the order of the task-decomposition weaving does not matter. Finally, the composition of *Payment Processor* aspectual element with the *Medi@* Actor needs also to bind |Goal 1 to *Get Bought Items* and |Task 4 to *Shopping Cart*. For the crosscut relationship properties of |Task4, we need to bind |when to after and |whom to none (this means after all sub-elements of *Shopping Cart* Task).

Notice also that the *how* property of this crosscut relationship is already stated as TD (Task Decomposition) because the relationship from a goal to a task can only be a task-decomposition link. As a result, in Figure 3, *Payment Processor* is composed with the *Item Transactor* aspectual element by adding a task-decomposition link from *Confirm Payment* goal to *Shopping Cart* task after all intentional elements. It is also composed with the *Medi@* actor by adding both a means-end link from *Confirm Payment* goal to *Get Bought Items* task and a means-end link from *Get Payment Information* goal to *Get Used Payment Way* goal.

3.4 Performing Trade-Off Analysis

After composing the aspectual elements with the i* models using the graphical composition rules, we should identify and resolve conflicting situations that may exist in composition points [30].

A trade-off analysis method could be considered, as for example [7], when we have two or more aspectual elements composed with the same element in a base module. We start by analyzing if these aspects influence negatively on each other. In such cases, we need to choose proper trade-off analysis methods to guide the conflict resolution.

In Figure 3, one conflicting situation could be identified in the *Manage Internet Shop* task at the *Medi@* actor. In this composition point three aspects, the *Security Manager*, *Availability Manager* and *Adaptability Manager*, are composed through a task-decomposition. Therefore, it is necessary to establish their order of composition.

In general, conflict resolution might lead to a revision of the requirements specification (stakeholders' requirements, aspectual requirements or composition rules). If this happens, the requirements are recomposed, the i* models are restructured and any further conflicts arising are resolved.

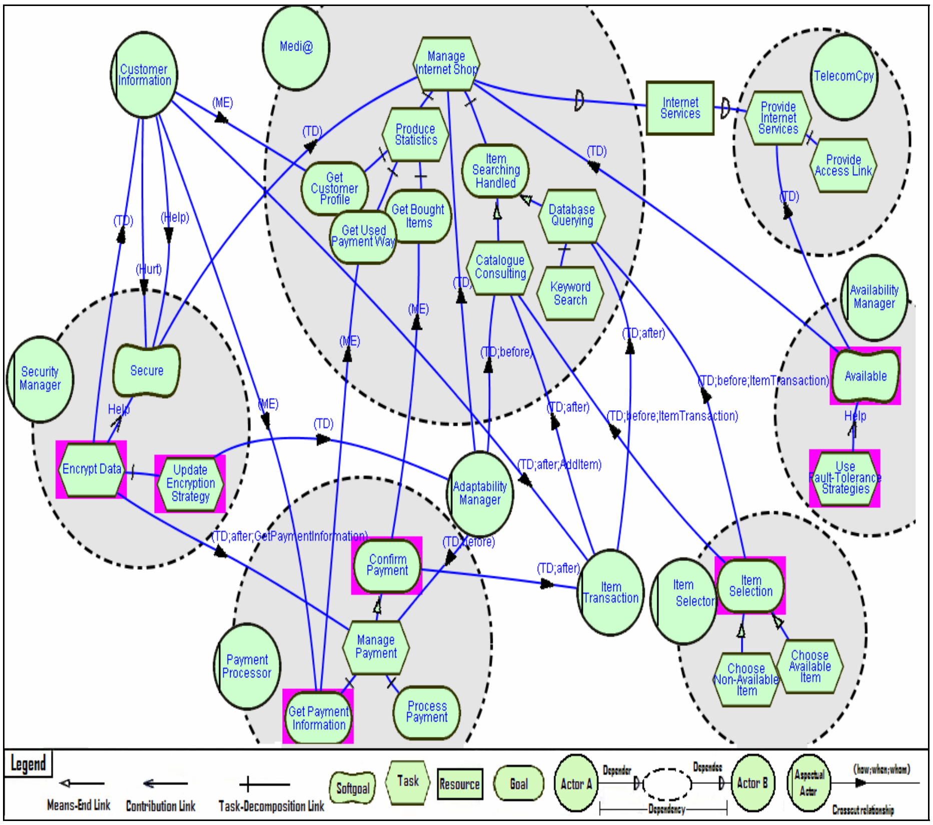


Fig. 3. The Medi@ Aspectual Strategic Rationale (SR) Model

4 Early Validation of Requirements Models

The V-model defines a software development process that supports incremental validation of software artefacts as well as code testing, according to a test-first perspective in software development that is becoming more and more compelling while the complexity of the system-to-be increases. In the V-model, validation and testing activities start at the beginning of the project, and complement requirements and design activities [12].

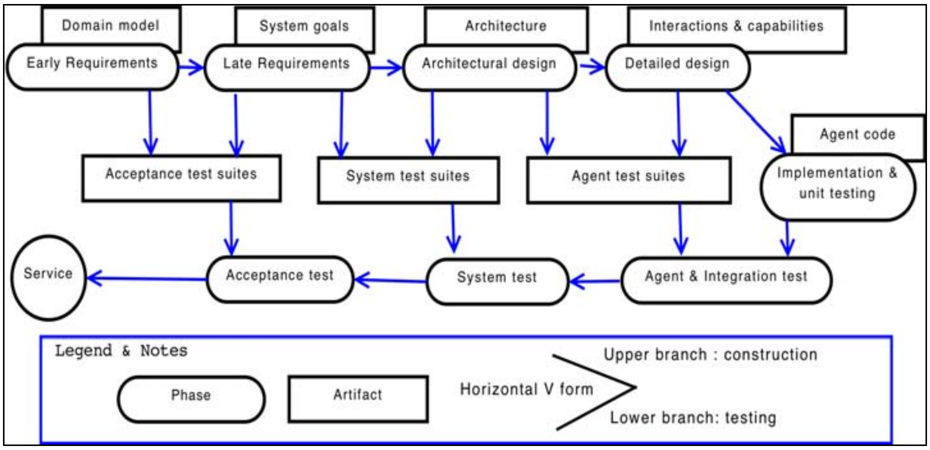


Fig. 4. The V-Model in GOST [26]

The Goal-Oriented Software Testing (GOST) approach proposed in [26], applies the V-model to the Tropos methodology [6]. In GOST test cases are derived from goal-oriented analysis and design models. GOST identifies five different validation and testing levels, each one addressing a specific objective, namely, acceptance, system, integration, agent, and unit testing. It provides also detailed procedures for deriving test suites from Tropos design artefacts, based on the relationship between design and testing artefacts depicted in Figure 4: the acceptance test's artefact is in relationship with early and late requirements models; system test with late requirements and architectural design models; agent test with architectural and detailed design; and unit testing is in relationship with detailed design and agent code.

In this section, we illustrate how the GOST approach works focusing on acceptance testing and show how we can derive test suites from early and late-requirements models using an excerpt of the Tropos requirements model of the *Medi@* system, depicted in Figure 5. Test cases for the other validation and testing levels can be derived following analogous procedures.

Acceptance testing aims at testing the software system in the customer execution environment (with the involvement of the stakeholders), and at verifying that the system meets the original stakeholder goals. In GOST, acceptance test suites (that is set of test cases) can be derived from early and late requirements models applying the following procedure[2]:

[2] The procedure to derive acceptance test suites has been here adapted since the original i* modelling language is used [37], instead of the Tropos variant described in [33].

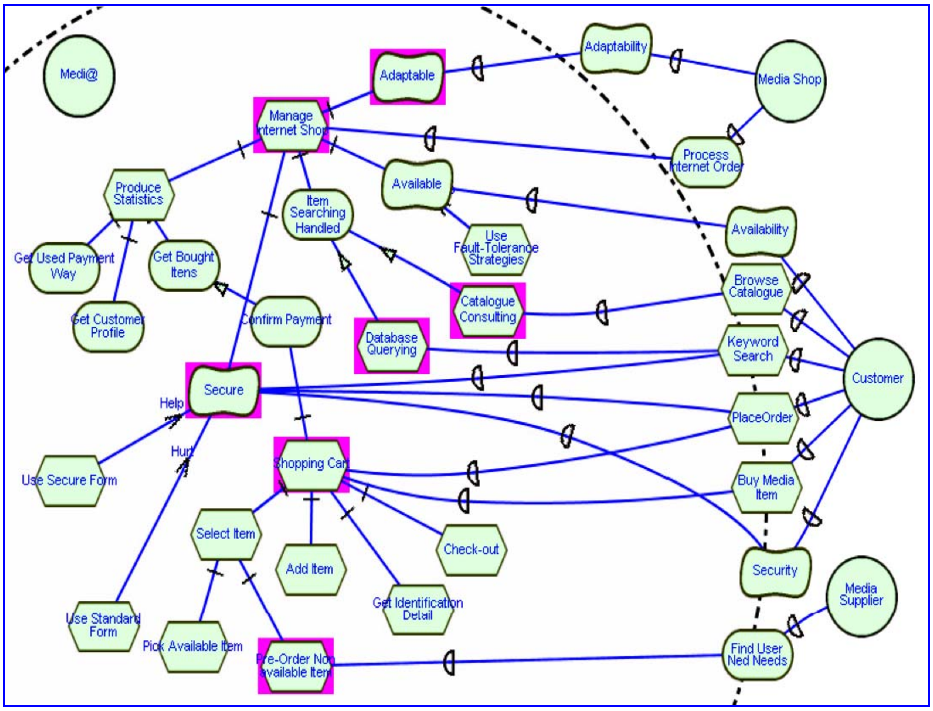


Fig. 5. An excerpt of the Medi@ Early and Late Requirements Model [10]

Acceptance test suites derivation consists of the following steps:

```
 1: forall actor ∈ {stakeholder actors}do
 2:    forall d ∈ {actor's dependencies towards the system}do
 3:       analyze the corresponding system goal/task/softgoal (in the SR model of the system)
 4:       for all lt ∈ {leaf task in the means-end / decomposition tree}
                      and sg ∈ {softgoal} do
 5:          /*create a test suite for lt and sg */
 6:          step1: identify operational or usage scenarios related to lt
 7:          step2: identify fulfillment criteria (oracle) for each scenario
 8:          step3: create one test suite with at least one test case for each scenario
 9:       endfor
10:    endfor
11: endfor
```

While deriving test cases we may discover underspecified or potentially conflicting situations that need to refine the original requirements specification. That is, the test-first perspective brings as additional benefits the possibility of preventing defects and faults and of improving requirements specifications [3].

This emerged also when applying the above procedure to the early- and late-requirements models of *Media@* which is partially reported in Figure 5, as discussed in the following.

Along step #2 in the test suite derivation procedure, we first identify the *Medi@* system requirements that derive from domain stakeholders dependencies. The result is illustrated in Table 2, which lists: the domain stakeholders (the *Media Supplier*, the *Customer* and the *Media Shop*); their dependencies to the system-to-be, namely the *Medi@* system; and the associated requirements that may be expressed in terms of goals, tasks, softgoals.

A first observation is that we have two different dependencies from the same domain stakeholder (the *Customer*) that define the same requirement of *Medi@*, which is represented by the *Shopping* task. This observation raises the following questions:

– is there any real difference between the task dependency and the goal dependency, or shall we consider the *Place Order* task as the intended way by the *Customer* to pursue its *Buy Media item* goal?
– why does the *Place Order* task dependency induce the *Secure* quality while the *Buy Media item* goal does not?

These questions should be posed back to the requirements analyst that can refine the model. A possible refinement could be that of considering only one of the two dependencies, maintaining the link to the two requirements expressed by the *Medi@*´s *Shopping Cart* task and *Secure* softgoal.

Let's focus now on the *Medi@*'s *Shopping Cart* task and apply steps from #4 to #9 to derive a test suite for it. Table 3, illustrates examples of test suites for the leaf tasks *Pick available item* and *Pre-Order non available item*, *Add Item* and *Check Out*[3].

Acceptance tests assume that an URL for *Medi@* is available for internet access through a web browser, and it serves a (set) of *Media Shop*(s) which sells DVD, Book, Video concerning a variety of categories, such as Sport, Music of different

Table 2. System-to-be Requirements Derived from Domain Stakeholders Dependencies.

Domain Stakeholder	*Dependency*	*Medi@ requirements*		
		Root Goal	*Root Task*	*Softgoal*
Media Shop	[G] Process Internet Order	-	Manage Internet Shop	-
	[SG] Adaptability	-	-	Adaptable
MediaSupplier	[G] Find User New Needs		Pre-Order Non Available Item	
Customer	[SG] Availability			Available
	[G] Buy Media Item	-	Shopping Cart	-
	[T] Place Order	-	Shopping Cart	Secure
	[T] Keyword Search	-	Database Querying	Secure
	[T] Browse Catalogue	-	Catalogue Consulting	
	[SG] Security	-		Secure

[3] The complete application of the steps #4-#9 to the Shopping cart task will derive test suites also for the other leaf tasks, namely Check Out, Get Identification Detail. They are not shown here for space reasons.

Table 3. Examples of Test Suite derived by applying the Acceptance Test Suite Derivation Procedure

TS	Leaf task	TC	Scenario	Oracle
TS1	Pick available item	TC1.1	Given a list of 4 items each one identified by a unique name (or a short description) the user can point and click on the name of the third item with the mouse or the pen-stick.	An instance of selected-item with the ID of item #3 is created and ready to be added to the cart.
		TC1.2	Given a list of 4 items, two having similar or equal names, the user can point it with the mouse (or the pen-stick) to get a short description. A further click on it will define its selection.	An instance of selected-item with the right ID is created and ready to be added to the cart.
		TC1.3	The available item list is empty	The customer can switch to the Pre-Order non available item function or perform another query
		TC1.4	The session expires while the customer pick an item from the list	Resuming the session the customer is informed about the current selected items
TS2	Pre-order non avail-able item	TC2.1	The customer can select an item marked as not available	An instance of selected-item with the right ID is created and ready to be added to the cart in the pre-order set.
TS3	Add Item	TC3.1	The customer add to the cart one item from the list of selected items (both available or not)	The cart set is updated upon the inclusion of the added item
		TC3.2	The customer add to the cart all the selected items (both available or not)	The cart set is updated upon the addition of the selected items
TS4	Check Out	TC4.1	The customer is shown the list of items put in the cart and confirm the order	The order of the selected items is ready to be completed with the customer payment info
		TC4.2	The session expires	The order info are saved and ready to be resubmitted to the customer for confirmation

genres. These *Media Shops* rest on (a) *Media Supplier(s)* to have the ordered items available to be sent to the customers. *Customers* access to the *Medi@* system with a laptop equipped with a mouse or with a PDA equipped with a pen-stick.

Focusing on TS2 we may notice that the Pre-order non available item leaf task results from the analysis of the *Medi@´*s *Shopping Cart* root task as well as from the dependency from the *Media Supplier* domain stakeholder to achieve the *Find User New Need* goal. The requirements model seems to assume that the Medi@ system is able to provide to the customers a list of items which fit their current needs, and are marked as available or not. This requires that the *Media Suppliers*, the *Media Shop* is working with, allow the system to access their product databases, which should be dynamically updated with respect to the (non)/availability of their products.

This requirement should be made explicit, for instance in the analysis of the *Item Searching Handled* goal, or in the associated *Database Querying* and *Catalogue Consulting* tasks. Here the database or the catalogue the two tasks refer to, may correspond to a distributed database (or catalogue) that can be built dynamically by accessing to the catalogues of the media suppliers that work for the *Media Shop*.

5 Dealing with Complexity Using a Combined Goal and Scenario Approach

The combined use of goals and scenarios has been explored within requirements engineering, primarily for eliciting, validating and documenting software requirements. Van Lamsweerde and Willement studied the use of scenarios for requirements elicitation and explored the process of inferring formal specifications of goals and requirements from scenario descriptions in [21].

In the CREWS project, Rolland et al. have proposed the coupling of goals and scenarios in requirements engineering with CREWS-L'Ecritoire [31]. In CREWS-L'Ecritoire, scenarios are used as a means to elicit requirements/goals of the system-to-be. Both goals and scenarios are represented as structured text. The coupling of goal and scenario could be considered as a "tight" coupling, as goals and scenarios are structured into <Goal, Scenario> pairs, which are called "requirement chunks". Their work focuses mainly on the elicitation of functional requirements/goals.

The Software Architecture Analysis Method (SAAM) [17] is a scenario-based method for evaluating architectures. It provides a means to characterize how well a particular architectural design responds to the demands placed on it by a particular set of scenarios. Based on the notion of context-based evaluation of quality attributes, scenarios are used as a descriptive means of specifying and evaluating quality attributes. SAAM scenarios are use-oriented scenarios, which are designed specifically to evaluate certain quality attributes of architecture. The evaluations are done using simulations or tests on a finished design.

This section first introduces the goal and scenario model integrated design process based on *Tropos* and UCM [8]. Then the running example is used to illustrate how to deal with design decision making problems with the Tropos concepts. Part of this work is based on [22] and the URN [17] notation, new development of this work is that we aim at using the joint goal and scenario analysis to cope with the complexity in the organizational environment – mutual social dependencies, conflicting intentions and interests, hard to express operational scenarios.

5.1 Coping with Complexity Using TROPOS

Based on the Tropos concepts, we now explore how to capture the organizational and environmental complexity with strategic dependency network, and how agents can respond to the complexity according to their own needs and capabilities based on strategic rationale analysis.

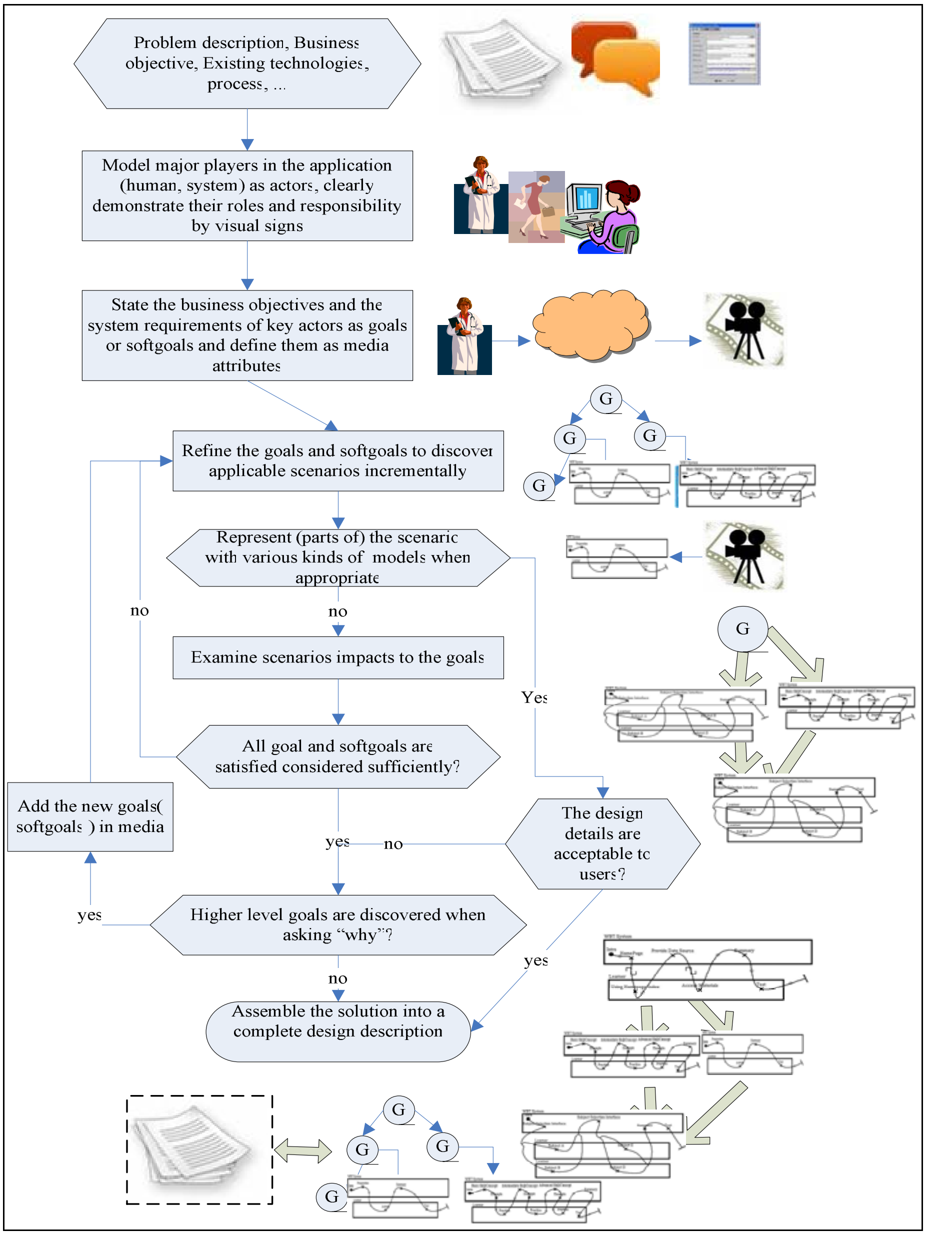

Problem description, Business objective, Existing technologies, process, ...
Model major players in the application (human, system) as actors, clearly demonstrate their roles and responsibility by visual signs
State the business objectives and the system requirements of key actors as goals or softgoals and define them as media attributes
Refine the goals and softgoals to discover applicable scenarios incrementally
Represent (parts of) the scenario with various kinds of models when appropriate
no
no
Examine scenarios impacts to the goals
Yes
All goal and softgoals are satisfied considered sufficiently?
Add the new goals(softgoals) in media
yes
no
The design details are acceptable to users?
yes
Higher level goals are discovered when asking "why"?
yes
no
Assemble the solution into a complete design description
G

Fig. 6. Goal and Scenario Model Integrated Design Process

We use the running example to illustrate the complementary application of Tropos and UCM. The approach is applicable to information systems in general, where there are conflicting goals and tradeoffs during design. Starting from the identification of the major stakeholders of the domain, we explain in sequence how to capture the original business objectives of the stakeholders, refine and operationalize these objectives into applicable design alternatives with *Tropos* and how to visualize and concretize some solutions with UCM.

Step 1: Placing system design within its broader social context, the proposed modeling approach can help to address the following questions systematically: Who are the major players in the business domain? What kinds of relationships exist among them? What are the business objectives and criteria of success for these players? The various dependency links in the model depict that in the Media example, the *Medi@* is a key player, who provides media products and services to *Customers* through the Internet. At the same time, it depends on the support of *Telecom Company, Media Supplier* and *Bank.*

Step 2: After the main players are identified, we ask them what their business objectives are, i.e., what they hope to accomplish for their organization, their sponsors, or their financial backers. Assume that, in our specific e-commerce system, the *Medi@* is playing the role of "Media Service Provider", who should then have two things in mind:

Attract new customer by selling media products online
Improve availability, adaptability and security of the service
They are represented as softgoals in the *Tropos* model in Figure 7.

Step 3: Explore the alternative business processes, methods or technologies used in this industry or business. Evaluate how these alternatives are serving the specific business objectives and the quality expectations of stakeholders.

In Figure 7, we see how the two solutions *Medi@* and Conventional *Media Shop* contribute differently to the goals. By using contribution links labeled with numbers or different symbolic types, the model portrayed that *Medi@* ***makes*** the goal of *Availability* satisficable, while *media shop* method ***hurts*** the fulfillment of this goal. Furthermore, the fulfilling of this goal ***helps*** the achievement of *Attract New Customer* goal. The result of this analysis suggests that *Medi@* may be a better option for current stakeholder. Part of this model (the two softgoals and the help relationship between them) is only applicable to current system, while other part (the structure showing the different resource consumption of the two solutions) depicts generic domain knowledge reusable to all service providers of *Media Products.*

Step 4: The advantages and disadvantages of the candidate solution are further investigated by evaluating its contributions to other concerned softgoals. For each disadvantage, mitigation plans are considered to complement the current solution.

The corresponding goal model shows that the advantages of *Media@* include *Availability*, *Increase Market Share* and *Adaptablity* is satisfied. Consequently, the overall quality of service improved. It also contributions positively to *Globalization*, *Flexibility*, both of which contribute positively (helps) to the customer's satisfactory.

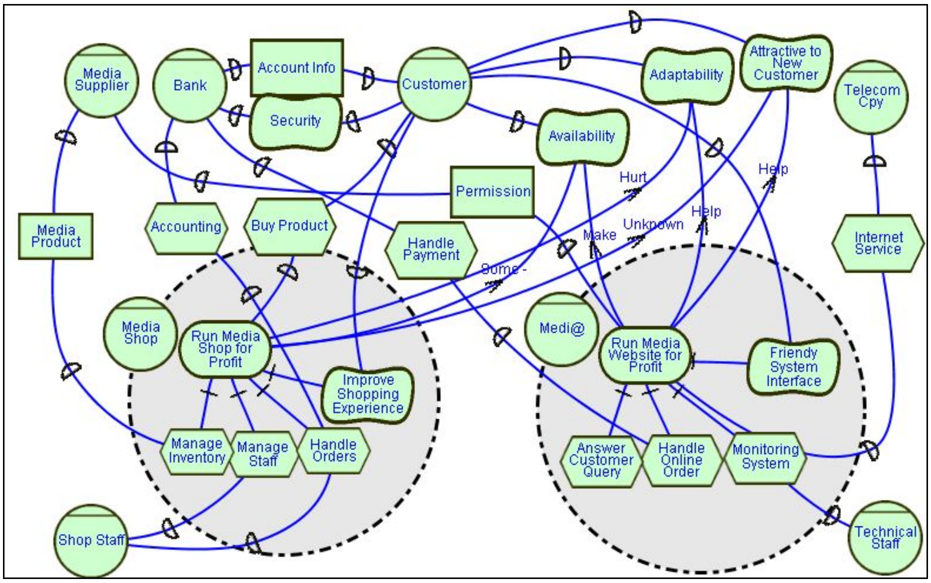


Fig. 7. Two Alternative Solutions in the Medi@ Example

However, there are also disadvantages e.g., the inherent *Security* and *More Efforts* on *Electronic Delivery* of *Media@* hurts the high level goals of the stakeholder. These disadvantages can be mitigated by countermeasures such as "DRM", which is represented as tasks connected with a negative correlation links (the dotted lines with arrows) to the unfavorable contributions links in the graph.

To identify the best design solutions, goal-reasoning techniques such as qualitative goal labeling algorithms are used. Quantitative techniques, such as probability or other quantitative measures, are used. With the help of *i** model, we are able to explore a space of design alternatives of considerable size. If there are *m* decision points (goals/softgoals with black rectangle shadow) and average *n* options at each point, there will be about n^m alternatives to be chosen from. Considering the presence of some external domain constraints, not all of alternatives are workable. When there is a large space of alternatives to choose from, system designers will greatly appreciate automated support such as an approximate ranking according to some criteria. The ranking of design alternatives is determined by the contributions to the softgoals of concern. In order to rank design alternatives, various criteria can be adopted. We can then either rank alternatives according to their overall contributions to all softgoals, or rank according to user's specific preferences.

Step 5: Identify the alternative essential sub-processes/components to implement the candidate solution. Next, we build model to elaborate the generic knowledge about *Media Shop Managed*. First of all, a media shop manager needs to *Choose a Business Front-end*, decide whether to use *bricks* or *clicks*, and *order handling Process* for the business. As all of these sub-processes are necessary steps for the finishing of the root task, they are represented as subgoals connected to the root task with decomposition links.

Step 6: As the goal-oriented design proceeds, finer-grained analysis needs to be conducted; hence the scenario-based notation comes into use. To elaborate the goal *Pick Ordering Process*, alternative processes are denoted in the *i** model as task nodes having different usage. For instance, *Media Shop* provides physical shopping experience, as they provide a *Safe* and *convenient* solution.

Each of the alternative processes can be described as a UCM scenario. *Medi@* system and *Customer* are represented as agent components (rectangles), holders of responsibilities (small crosses along on the wiggle lines). In the scenario, the use case path shows that different *Customers* can have different routines if they choose different subjects in the Web Interface. The *Customer* and the *Medi@* system collaborates on searching on the web for materials of interest, so they are sharing responsibilities (denoted by adding a square *S* between the shared responsibilities).

Having analyzed the benefits and tradeoffs of these structures, we can see that UCM is a useful counterpart to Tropos in the process from requirements to high-level design, because it provides a concrete model of each design alternative. Based on the features in such a model, new non-functional requirements may be detected and added to the Tropos model. At the same time, in the Tropos model, new means to achieve the functional requirements can always be explored and concretized in a UCM model. Thus the above design process may iterate several rounds until an acceptable design is made.

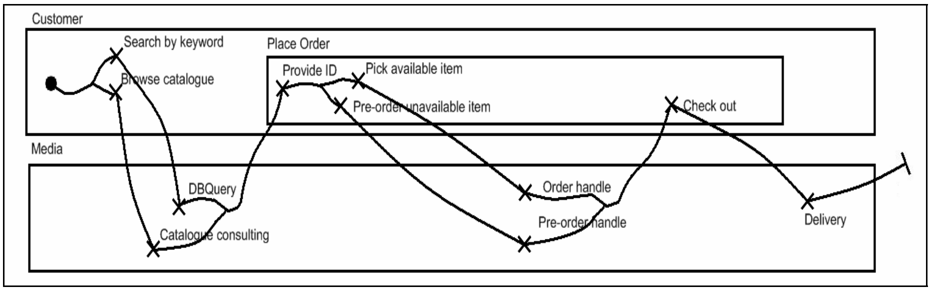


Fig. 8. Medi@ Scenario Model in UCM

6 Software Project Management Process

Due to benefits and perspectives such as efficient software project management, continuous organizational modeling and requirements acquisition, early implementation, continuous testing and modularity, iterative development is more and more used by software engineering professionals especially through methodologies such as the *Unified Process* [34].

Most agent-oriented software development and requirements-driven methodologies only use a waterfall system development life cycle (SDLC) or advice their users to proceed iteratively without offering a strong project management framework to support that way of proceeding. Consequently they are not suited for the development of huge and complex user-intensive applications. The aim of this section is to present a research dedicated to extend *Tropos* with an iterative life cycle called *I-Tropos*[4]. This

[4] *I-Tropos* stands for *Iterative Tropos*.

methodology fills the project and product life cycle gaps of Tropos and offers a goal-oriented project management perspective to support project stakeholders for applying the methodology.

The *I-Tropos* project management framework covers several dimensions including risk, quality, time and process management. Contributions include, among others, taking threats and quality factors' evaluation directly in account for planning the goals realization over multiple iterations. The process is exposed in this section and illustrated on the *Medi@* case study using *DesCARTES*, a CASE-tool designed to support I-Tropos.

6.1 Process Engineering Concepts

An *I-Tropos development* is made of disciplines[5] iteratively repeated while the relative effort spend on each one is variable from one iteration to the other. The Organizational Modeling and Requirements Engineering disciplines are respectively largely inspired by Tropos' Early and Late Requirements disciplines. The Architectural and Detailed Design disciplines correspond to the same stages of traditional Tropos. *I-Tropos* includes core activities i.e. *Organizational Modeling*, *Requirements Engineering*, *Architectural Design*, *Detailed Design*, *Implementation*, *Test* and *Deployment* but also support disciplines to handle the project development called *Risk Management*, *Time Management*, *Quality Management* and *Software Process Management*. There is little need for support activities in a process using a waterfall SDLC since the core disciplines are sequentially achieved one for all. When dealing with a process using an iterative SDLC, the need for support disciplines for managing the whole software project is from primary importance. I-Tropos process' disciplines are described in detail in [35].

Using an iterative SDLC implies repeating process' disciplines many times during the software project. Each iteration belongs to one of the four phases inspired by the Unified Process (UP); a complementary documentation can be found in [20] while a summary of each phase objective is depicted into the next section. These phases are achieved sequentially and have different goals evaluated at milestones through knowledge and achievement oriented metrics, those are informally described into the next section. Figure 9 offers a two dimensional view of the I-Tropos process: it shows the disciplines and the four different phases they belong to.

[5] The phase and discipline notions are often presented as synonyms in software engineering literature. In [24], Tropos is described as composed of five phases (Early Requirements, Late Requirements, Architectural Design, Detailed Design and Implementation). However [29] defines disciplines as "*a particular specialization of Package that partitions the Activities within a process according to a common "theme".*", while the phase is defined as "*a specialization of WorkDefinition such that its precondition defines the phase entry criteria and its goal (often called a "milestone") defines the phase exit criteria*". In order to be compliant with the most generic terminology, traditional Tropos phases will be called disciplines in our software process description since they partition Activities under a common theme. In the same way, phases will be considered as groups of iterations which are workflows with a minor milestone.

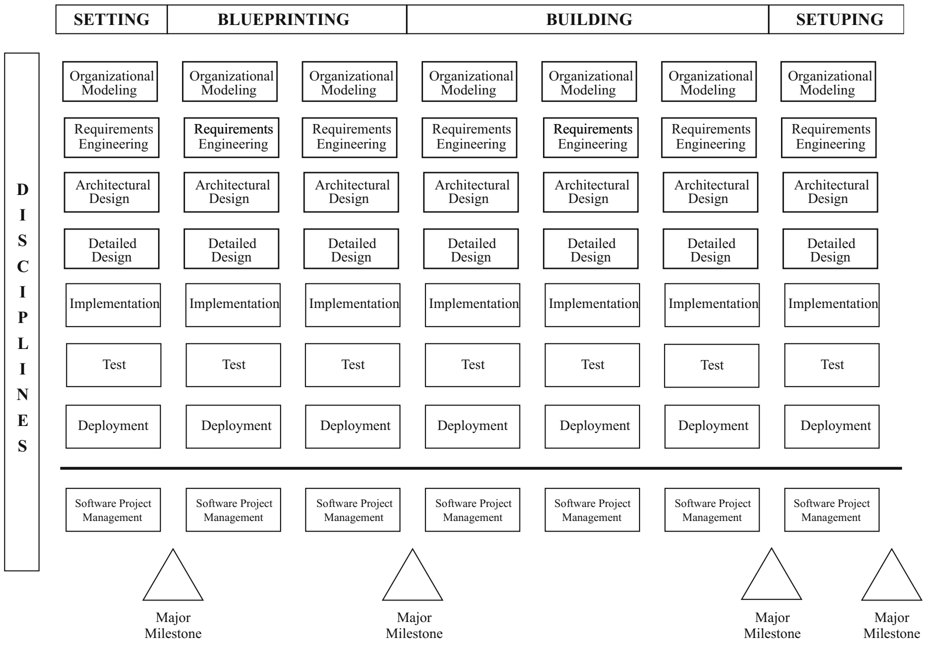


Fig. 9. *I-Tropos*: Iterative Perspective

6.2 Process Phases

I-Tropos phases are inspired by the UP phases[6]; each one is made of one or more iterations. Disciplines are gone through sequentially; as stressed before the phases are separated by major milestones. Each of them has its own goal:

The *setting* phase is designed to identify and specify most stakeholders requirements, have a first approach of the environment scope, identify and evaluate project's threats and identify and evaluate quality factors.

The *blueprinting* phase is designed to produce a consistent architecture for the system on the basis of the identified requirements, eliminate most risky features in priority and evaluate blueprints/prototypes to stakeholders; feedback will feed next iterations.

The *building* phase is designed to build a working application and validate developments.

The *setuping* phase is designed to finalize production, train users and document the system.

6.3 Process Core Disciplines

The *I-Tropos* process has been fully described using the Software Engineering Process Metamodel in [35]. That technical report describes each process' *Discipline*, *Activity*,

[6] The phases milestones expressed hereafter are based on the metrics expressed in the Unified Process (see [20]).

Role, *WorkDefinition* and *WorkProduct*, so that it can be used as reference or guide to the methodology. A lightened overview of the process is given in this section.

The Organizational Modeling discipline, strongly inspired from the Tropos *Early Requirements* stage, aims to understand the problem by studying the existing organizational setting.

The Requirements Engineering discipline, inspired from the Tropos *Late Requirements* stage, extends models created previously by including the system to-be, modeled as one or more actors.

The Architectural Design discipline, inspired by the Tropos *Architectural Design* stage, aims to build the system's architecture specification, by organizing the dependencies between the various sub-actors identified so far, in order to meet functional and non-functional requirements of the system.

The Detailed Design discipline, inspired by Tropos *Detailed Design*, aims at defining the behavior of each architectural component in further detail.

The Implementation discipline aims to produce an executable release of the application on the basis of the detailed design specification.

The Test discipline aims on evaluating the quality of the executable release.

The Deployment discipline aims to test the software in its final operational environment.

6.4 Process Support Disciplines

These disciplines provide features to support software development i.e. tools to manage risks, quality levels, time, resources allocation but also the software process itself. All those features can be regrouped onto the term *software project management*.

Risk Management is the process of identifying, analyzing, assessing risk as well as developing strategies to manage it. Strategies include transferring risk to another party, avoiding risk, reducing its negative effects or accepting some or all of the consequences of a particular one. Technical answers are available to manage risky issues. Choosing the right mean to deal with particular risk is a matter of compromise between level of security and cost. This compromise requires an accurate identification of the threats as well as their adequate evaluation.

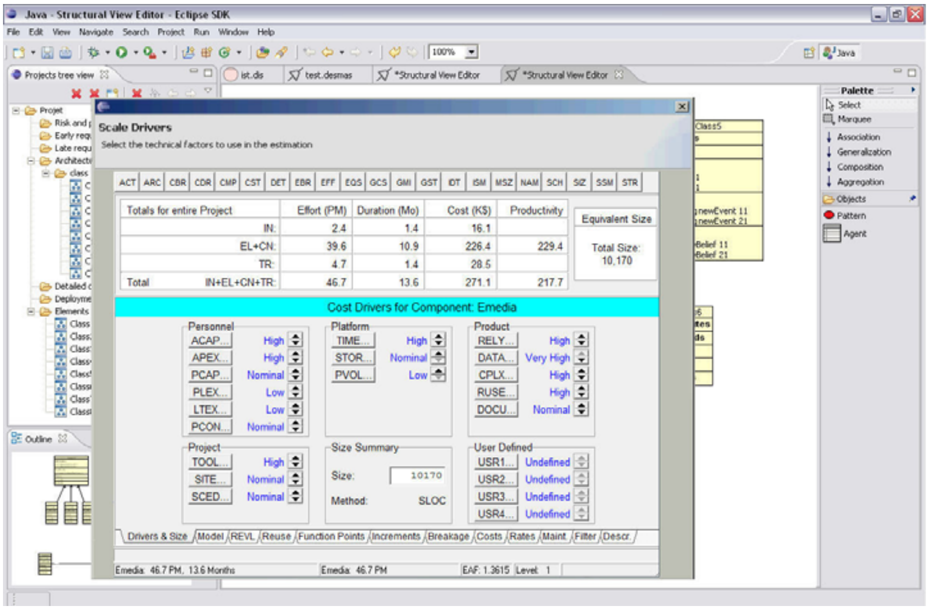
Quality Management is the process of ensuring that quality expected and contracted with clients is achieved throughout the project. Strategies include defining quality issues and the minimum quality level for those issues. Technical answers are available to reach quality benchmarks. Choosing the right mean to deal with quality issues is a matter of compromise between level of quality and cost. This compromise requires an accurate identification of the quality benchmarks as well as their adequate evaluation.

Time Management is the process of monitoring and controlling the resources (time, human and material) spent on the activities and tasks of a project. This discipline is of primary importance since, on the basis of the risk and quality analyses, the global iterations time and human resources allocation are computed; they are revised during each iteration.

Software Process Management is the use of process engineering concepts, techniques, and practices to explicitly monitor, control, and improve the systems engineering process. The objective of systems engineering process management is to enable an organization to produce system/segment products according to plan while simultaneously improving its ability to produce better products [9]. In this context, *Software Process Management* regroups the activities aimed to tailor the generic process onto a specific project as well as improving the software process.

6.5 Applying I-Tropos on Medi@

Figure 10 depicts *DesCARTES* [11], more specifically the cost and effort estimation interface provided by the module supporting the Software Project/Time Management Disciplines from *I-Tropos*. Project Management Features such as scale drivers, cost factors, increment settings, labor rates, breakage, etc., can directly be tuned through this kind of interfaces.



Java - Structural View Editor - Eclipse SDK
File Edit View Navigate Search Project Run Window Help
Projects tree view
ist.dis
test.desmas
*Structural View Editor
*Structural View Editor
Palette
Select
Marquee
Association
Generalization
Composition
Aggregation
Objects
Pattern
Agent
Outline
ACT ARC CBR CDR CMP CST DET EBR EFF EQS GCS GMI GST IDT ISM MSZ NAM SCH SIZ SSM STR
Drivers & Size / Model / REVL / Reuse / Function Points / Increments / Breakage / Costs / Rates / Maint / Filter / Descr.

Scale Drivers

Select the technical factors to use in the estimation

Totals for entire Project		Effort (PM)	Duration (Mo)	Cost (K$)	Productivity
	IN:	2.4	1.4	16.1	
	EL+CN:	39.6	10.9	226.4	229.4
	TR:	4.7	1.4	28.5	
Total	IN+EL+CN+TR:	46.7	13.6	271.1	217.7

Equivalent Size

Total Size: 10,170

Cost Drivers for Component: Emedia

Personnel		Platform		Product	
ACAP...	High	TIME...	High	RELY...	High
APEX...	High	STOR...	Nominal	DATA...	Very High
PCAP...	Nominal	PVOL...	Low	CPLX...	High
PLEX...	Low			RUSE...	High
LTEX...	Low			DOCU...	Nominal
PCON...	Nominal				

Project		Size Summary		User Defined	
TOOL...	High	Size:	10170	USR1...	Undefined
SITE...	Nominal	Method:	SLOC	USR2...	Undefined
SCED...	Nominal			USR3...	Undefined
				USR4...	Undefined

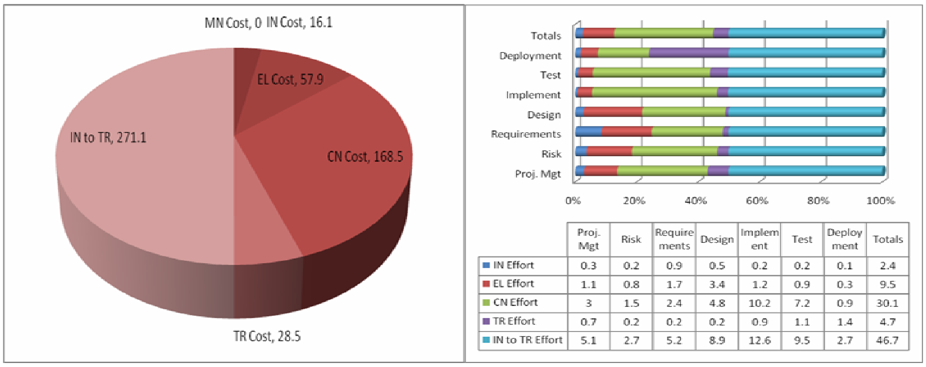
Emedia: 46.7 PM, 13.6 Months Emedia: 46.7 PM EAF: 1.3615 Level: 1

Fig. 10. *DesCARTES*: Estimating the Medi@ Application with the I-Tropos Software Project/Time Management Disciplines Module

DesCARTES (Design CASE Tool for Agent-Oriented Repositories, Techniques, Environments and Systems) Architect is a Computer-Aided Software Engineering Tool developed as a plug-in for the Eclipse Platform by the Information Systems Unit at the University of Louvain. It is designed to support various models edition: i* models (Strategic Dependency and Strategic Rationale models), NFR models, UML models, AUML models in the context of *I-Tropos* or *Unified Process*-like developments. The originality of the tool is that it allows the development of the methodology models throughout iterative software life cycle processes as well as forward engineering capabilities and integrated software project management, time and risk/quality management modules.

Figure 11 provides graphical reporting outputs directly produced by *DesCARTES* related to the cost, effort, activities and schedule estimation for Medi@.

Instantiated to Medi@, these outputs applying regression models based on COCOMO or SLIM [4] and factor scales supported by maturity models such as CMM-I give the following estimation figures. Total size is estimated to 101700 Java SLOC while the total cost will be 271 400 $. The duration of the project will be 13.6 months with 46.7 actual person-months (PM) and a nominal PM at 29.1. Productivity is estimated to 256.9 SLOC per PM at the unit cost at 22.26$ per line. Average staffing during the project is 3.43 persons with a high at 4.87 during *Building* a low at 1.71 during *Setting*.



	Proj. Mgt	Risk	Requirements	Design	Implement	Test	Deployment	Totals
🟦 IN Effort	0.3	0.2	0.9	0.5	0.2	0.2	0.1	2.4
🟥 EL Effort	1.1	0.8	1.7	3.4	1.2	0.9	0.3	9.5
🟩 CN Effort	3	1.5	2.4	4.8	10.2	7.2	0.9	30.1
🟪 TR Effort	0.7	0.2	0.2	0.2	0.9	1.1	1.4	4.7
🟦 IN to TR Effort	5.1	2.7	5.2	8.9	12.6	9.5	2.7	46.7

Fig. 11. Graphical Outputs from *DesCARTES*: Cost, Effort, Activities and Schedule Estimation for Medi@

7 Conclusion

This chapter presents a set of approaches to deal with complexity, which address various activities in software development, namely requirements modeling, testing and project management.

More specifically, we outlined an approach to improve modularization of requirements models described in i*, by identifying, separating and composing crosscutting concerns. A specific notation has been created to represent aspectual *i** models. This leads to the addition of two new concepts in the *i*/Tropos* modeling language, namely aspectual element and crosscut relationship. Aspectual elements modularize crosscutting concerns while the crosscut relationship captures the information of source and target model elements, as well as, when and how an aspectual element crosscuts other model elements. The approach introduces modularity (it creates units that are strongly cohesive and loosely coupled), reduces the scalability (removing the redundant elements and links) and improves the reusability. Work in under way to evaluate the resulting models by means of metrics to assess well-known attributes in software engineering, such as separation of concerns, size, cohesion and coupling. In the near future we plan to define a trade-off analysis method to complement the proposed process as well as to provide tool support for the approach.

The application of a V-model software development process in Tropos, namely the GOST methodology [26], has been introduced, as an approach to enable incremental validation and testing of artifacts while building complex software system. Further benefits of using this test-first perspective for clarifying ambiguities in requirements models has also been illustrated by applying GOST to a fragment of the early- and late-requirements models of the *Media@* system. The systematic application of the GOST approach can be supported by the eCAT tool, which automatically generates test suite skeletons from goal models [25]. Extensions of GOST with automated test case generation techniques are described in [27] and [28]. This is a necessary step towards supporting a continuous validation and testing approach for the development of complex software systems.

The combined use of Tropos and UCM enables the description of functional and non-functional requirements, abstract requirements and concrete system models, intentional strategic design rationales and non-intentional details of concurrent, temporal aspects of the future system. It is natural to adopt Tropos as a basic requirements knowledge representation language, and try to find how other existing requirements modeling languages relate and complement to it. So following the attempt in integrating i*(GRL) with UCM, we move to integrate *i** with the Problem Frames. The ultimate objective is to build a requirement ontology that incorporates as many perspectives as possible.

In terms of software process management, *I-Tropos* represents an evolution of the Tropos process. It constitutes an operationalization of the Tropos methodology in order to be used in large software developments. *I-Tropos* mainly fills up the gap of project management which is, for now, seldom approached in MAS literature. The main contributions include meta-level process documentation, a full software and product life cycle coverage, a project management framework for the inclusion of Tropos developments into an iterative and incremental process template and the support of a specific CASE tool, *DesCARTES*. *I-Tropos* is an adequate project management for building large-scale enterprise systems from scratch. However, many firms have turned to the reuse of existing software or using commercial off-the-shelf (COTS) software as an option due to lower cost and time of development. Work for enlarging its scope including COTS software customization onto specific case and adequate project management is in progress. The basic adaptation of *I-Tropos* to support this paradigm of software development is described in [36].

Acknowledgements

This work was supported by several research grants: National Natural Science Foundation of China (Grant No.60873064), National Basic Research and Development 973 Program of China (Grant No.2009CB320706), CNPq Grant 308587/2007-3, BIT Initiative, FUR-PAT Trento (STAMPS project). We thank our colleagues at Universidade Federal de Pernambuco and Universidade Nova de Lisboa, in particular Fernanda Alencar, João Araújo and Ana moreira for their contributions to the aspectualization of i* models.

References

1. Amyot, D.: Introduction to the User Requriements Notation: Learning by Example. Computer Networks~42(3), 285--301 (2003)
2. Alencar, F., Castro, J., Moreira, A., Araújo, J., Silva, C., Ramos, R., Mylopoulos, J.: Integration of aspects with i* models. In: Kolp, M., Henderson-Sellers, B., Mouratidis, H., Garcia, A., Ghose, A.K., Bresciani, P. (eds.) AOIS 2006. LNCS, vol. 4898, pp. 183–201. Springer, Heidelberg (2008)
3. Beck, K.: Test Driven Development: By Example. Addison-Wesley Longman Publishing Co., Inc., Boston (2002)
4. Boehm, B., et al.: Software cost estimation with COCOMO II. Prentice-Hall, Englewood Cliffs (2000)
5. Bresciani, P., Perini, A., Giunchiglia, F., Giorgini, P., Mylopoulos, J.: A Knowledge Level Software Engineering Methodology for Agent Oriented Programming. In: Proc. of the 5th Int. Conference on Autonomous Agents (Agents 2001), Montreal, pp. 648–655 (2001)
6. Bresciani, P., Giorgini, P., Giunchiglia, F., Mylopoulos, J., Perini, A.: Tropos: An Agent-Oriented Software Development Methodology. Autonomous Agents and Multi-Agent Systems 8(3), 203–236 (2004)
7. Brito, I.S., Vieira, F., Moreira, A., Ribeiro, R.A.: Handling conflicts in aspectual requirements compositions. In: Rashid, A., Aksit, M. (eds.) Transactions on AOSD III. LNCS, vol. 4620, pp. 144–166. Springer, Heidelberg (2007)
8. Buhr, R.J.A.: Use Case Maps as Architectural Entities for Complex Systems. Transactions on Software Engineering 24(12), 1131–1155 (1998)
9. Capability Assessment Working Group on Systems Engineering, Systems Engineering Capability Assessment Model, INCOSE-TP-1996-002-01, Version 1.50a (June 1996)
10. Castro, J., Kolp, M., Mylopoulos, J.: Towards Requirements-Driven Information Systems Engineering: The Tropos Project. Information Systems Journal 27(6), 365–389 (2002)
11. DesCARTES Architect: Design CASE Tool for Agent-Oriented Repositories, Techniques, Environments and Systems (2008), `http://www.isys.ucl.ac.be/descartes/`
12. Development Standards for IT Systems of the Federal Republic of Germany, The V-Model (2005), `http://www.v-modell-xt.de`
13. Estrada, H., Rebollar, A.M., Pastor, Ó., Mylopoulos, J.: An empirical evaluation of the i* framework in a model-based software generation environment. In: Dubois, E., Pohl, K. (eds.) CAiSE 2006. LNCS, vol. 4001, pp. 513–527. Springer, Heidelberg (2006)
14. Feiler, P., Gabriel, R.P., Goodenough, J., Linger, R., Longsta, T., Sullivan, K., Wallnau, K.: Ultra-large-scale systems: The software challenge of the future. Technical report, Software Engineering Institute (July 2006), `http://www.sei.cmu.edu/uls/`
15. France, F., Kim, D., Ghosh, S., Song, E.: A UML-Based Pattern Specification Technique. IEEE Transactions on Software Engineering 30(3), 193–206 (2004)
16. Graham, D.R.: Requirements and testing: Seven missing-link myths. IEEE Software 19(5), 15–17 (2002)
17. International Telecommunications Union (ITU-T) Recommendation Z.151: User Requirements Notation (URN) - Language Definition (2008)

18. Kazman, R., Bass, L., Abowd, G., Webb, M.: SAAM: A Method for Analyzing the Properties of Software Architectures. In: Proceedings of the 16th International Conference on Software Engineering, Sorrento, Italy, May 1994, pp. 81–90 (1994)
19. Kim, D., France, R., Ghosh, S., Song, E.: Using Role-Based Modeling Language as Precise Characterizations of Model Families. In: 8th Intl. Conf. on Engineering of Complex Computer Systems, IEEE, USA (2002)
20. Kruchten, P.: The Rational Unified Process: An Introduction. Addison-Wesley, Reading (2003)
21. van Lamsweerde, A., Willemet, L.: Inferring Declarative Requirements Specifications from Operational Scenarios. IEEE Transactions on Software Engineering, Special Issue on Scenario Management (December 1998)
22. Liu, L., Yu, E.: Designing Information Systems in Social Context: A Goal and Scenario Modelling Approach. Information Systems 29(2), 187–203 (2004)
23. Luck, M., McBurney, P., Shehory, O., Willmott, S.: Agent Technology: Computing as Interaction (A Roadmap for Agent Based Computing). AgentLink, Liverpool, UK (2005)
24. Mylopoulos, J., Castro, J.: Tropos: A Framework for Requirements-Driven Software Development. In: Brinkkemper, S., Lindencrona, E., Sølvberg, A. (eds.) Information Systems Engineering: State of the Art and Research Themes, pp. 261–273. Springer, Heidelberg (2000)
25. Nguyen, C.D., Perini, A., Tonella, P.: eCAT: a Tool for Automating Test Cases Generation and Execution in Testing Multi-Agent Systems (Demo Paper). In: Proc. Of the 7th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2008), Demo Proceedings, Estoril, Portugal, May 12-16, 2008, pp. 1669–1670 (2008)
26. Nguyen, C.D., Perini, A., Tonella, P.: Goal-Oriented Testing for MAS. Int. Journal of Agent-Oriented Software Engineering (submitted, 2008)
27. Nguyen, C.D., Perini, A., Tonella, P.: Automated Continuous Testing of Autonomous Distributed Systems. In: 1st International Workshop on Search-Based Software Testing, in conjunction with the IEEE International Conference on Software Testing, Verification and Validation, ICST 2008 (2008)
28. Nguyen, C.D., Miles, S., Perini, A., Tonella, P., Harman, M., Luck, M.: Evolutionary Testing of Autonomous Software Agents. In: Decker, Sichman, Sierra, Castelfranchi (eds.) Proc. of 8th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2009), Budapest, Hungary, May 10-15 (2009)
29. OMG: The Software Process Engineering Metamodel Specification. Version 1.1 (January 2005)
30. Rashid, A., Moreira, A., Araujo, J.: Modularisation and Composition of Aspectual Requirements. In: Proc. of the 2nd Intl. Conf. on Aspect-Oriented Software Development, pp. 11–20. ACM Press, New York (2003)
31. Rolland, C., Grosz, G., Kla, R.: Experience With Goal-Scenario Couplingin Requirements Engineering. In: Proceedings of the IEEE International Symposium on Requirements Engineering 1998, Limerick, Ireland (1998)
32. Silva, C., Araújo, J., Moreira, A., Castro, J.: Designing Social Patterns using Advanced Separation of Concerns. In: Krogstie, J., Opdahl, A.L., Sindre, G. (eds.) CAiSE 2007 and WES 2007. LNCS, vol. 4495, pp. 309–323. Springer, Heidelberg (2007)

33. Susi, A., Perini, A., Giorgini, P., Mylopoulos, J.: The Tropos metamodel and its use. Informatica 29(4), 401–408 (2005)
34. Royce, W.: Software Project Management. A Unified Framework. Addison-Wesley, Reading (1998)
35. Wautelet, Y., Kolp, M., Achbany, Y.: S-Tropos: An Iterative SPEM-Centric Project Management Process. Working Paper IAG 06/01, Université catholique de Louvain (2006)
36. Wautelet, Y., Achbany, Y., Kiv, S., Kolp, M.: A Service-Oriented Framework for Component-Based Software Development: An i* Driven Approach. In: Proceedings of the 11th International Conference on Enterprise Information Systems, ICEIS 2009, Milan (2009)
37. Yu, E.: Modelling Strategic Relationships for Process Reengineering. Ph.D Thesis, Department of Computer Science, University of Toronto, Canada (1995)

On Non-Functional Requirements in Software Engineering

Lawrence Chung[1] and Julio Cesar Sampaio do Prado Leite[2]

[1] Department of Computer Science, The University of Texas at Dallas
[2] Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro
`www.utdallas.edu/~chung/, www.inf.puc-rio.br/~julio`

Abstract. Essentially a software system's utility is determined by both its functionality and its non-functional characteristics, such as usability, flexibility, performance, interoperability and security. Nonetheless, there has been a lop-sided emphasis in the functionality of the software, even though the functionality is not useful or usable without the necessary non-functional characteristics. In this chapter, we review the state of the art on the treatment of non-functional requirements (hereafter, NFRs), while providing some prospects for future directions.

Keywords: Non-functional requirements, NFRs, softgoals, satisficing, requirements engineering, goal-oriented requirements engineering, alternatives, selection criteria.

1 Introduction

"Soft is harder to deal with than hard." [Anonymous]

Essentially a system's utility is determined by both its functionality and its non-functional characteristics, such as usability, flexibility, performance, interoperability and security. Nonetheless, there has been a lop-sided emphasis in the functionality of the system, even though the functionality is not useful or usable without the necessary non-functional characteristics.

Just with almost everything else, the concept of quality is also fundamental to software engineering, and both functional and non-functional characteristics must be taken into consideration in the development of a quality software system. However, partly due to the short history behind software engineering, partly due to the demand on quickly having running systems fulfilling the basic necessity, and also partly due to the "soft" nature of non-functional things, most of the attention in software engineering in the past has been centered on notations and techniques for defining and providing the functions a software system has to perform.

A frequently observable practice, as a result of this lop-sided emphasis in the functional side of a software artifact, is that the needed quality characteristics are treated only as technical issues related mostly to the detailed design or testing of an implemented system. This kind of practice, of course, is quite inadequate. Detailed design and testing do not make much sense without their preceding phases of understanding what the real-world problem is to which a software system might be proposed as a solution and also what the specifics of the software solution, i.e., the requirements, might be like. And real-world problems are more non-functionally oriented than they are functionally oriented, e.g., poor productivity, slow processing, high cost, low quality, and unhappy customer.

Although the requirements engineering community has classified requirements as either functional or non-functional, most existing requirements models and requirements specification languages lacked a proper treatment of quality characteristics. Treating quality characteristics as a whole, and not just as functionality alone, has been a key focus of works in the area of goal-oriented requirements engineering [1] [2], and in particular the NFR Framework [3] that treats non-functionality at a high level of abstraction for both the problem and the solution.

This chapter brings forth a review of the literature on NFRs, with emphasis in the different definitions, representation schemes, as well as more advanced uses of the concepts. At the end, we conclude the chapter by discussing open issues in the early treatment of NFRs and its impacts on software construction.

2 What are Non-Functional Requirements?

In literature, a plethora of definitions can be found of non-functional requirements (NFRs).

Colloquially speaking, NFRs have been referred to as “-ilities” (e.g., usability) or “-ities” (e.g., integrity), i.e., words ending with the string “-ility” or “-ity. A large list of such words can be found, for example, in [3]. There are many other types of NFRs that do not end with either “-ility” or “-ity” as well, such as performance, user-friendliness and coherence.

An important piece of work on NFRs is the NFR Framework [1] [3], which decouples the concept of functionality from other quality attributes and concerns for productivity, time and cost, by means of a higher-level of abstraction. Instead of focusing on expressing requirements in terms of detailed functions, constraints and attributes, the NFR Framework devised the distinction of NFRs by using the concepts of goal and softgoal. More details on the NFR Framework will be described further in Section 4.

In the area of Software Architecture, one frequently encountered keyword is “quality attributes” [4], which is understood as a set of concerns related to the concept of quality. For a definition of quality, an IEEE standard [5] is used here as a companion: “Software quality is the degree to which software possesses a desired combination of attributes (e.g., reliability, interoperability).”

Several other authors have also treated these types of concerns. For instance, basic quality (functionality, reliability, ease of use, economy and safety) is distinguished from extra quality (flexibility, reparability, adaptability, understandability, documentation and enhanceability) in [6].

In the area of engineering and management, the well known QFD (Quality Function Deployment) strategy [7] distinguishes positive quality from negative quality: “QFD is quite different in that it seeks out both "spoken" and "unspoken" customer requirements and maximizes "positive" quality (such as ease of use, fun, luxury) that creates value. Traditional quality systems aim at minimizing negative quality (such as defects, poor service)”. One of the techniques used by QFD strategies is the House of Quality [8], in which the process starts "...with the customer, whose requirements are called customer attributes (CA´s) - phrases customers use to describe products and product characteristics...". Incidentally, none of the examples of the CA´s in [8] is related to functionality or just functionality alone.

In the area of software requirements, the term non-functional requirements [9] has been used to refer to concerns not related to the functionality of the software. However, different authors characterize this difference in informal and unequal definitions. For example, a series of such definitions is summarized in [10]:

a) “Describe the non-behavioral aspects of a system, capturing the properties and constraints under which a system must operate. “

b) “The required overall attributes of the system, including portability, reliability, efficiency, human engineering, testability, understandability, and modifiability.”

c) “Requirements which are not specifically concerned with the functionality of a system. They place restrictions on the product being developed and the development process, and they specify external constraints that the product must meet.”

d) “... global requirements on its development or operational cost, performance, reliability, maintainability, portability, robustness, and the like. ... There is not a formal definition or a complete list of nonfunctional requirements.”

e) “The behavioral properties that the specified functions must have, such as performance, usability.”

f) “A property, or quality, that the product must have, such as an appearance, or a speed or accuracy property.”

g) “A description of a property or characteristic that a software system must exhibit or a constraint that it must respect, other than an observable system behavior.”

After arguing that the definitions are unclear and that they lack consensus, the author says: “For persons who do not want to dispose of the term ‘non-functional requirement’, we can define this term additionally as: DEFINITION. A non-functional requirement is an attribute of or a constraint on a system.” [10].

There are other additional definitions in literature worth adding to the list.

h) ” … types of concerns: functional concerns associated with the services to be provided, and nonfunctional concerns associated with quality of service – such as safety, security, accuracy, performance, and so forth.” [2].

i) “The term “non-functional requirement” is used to delineate requirements focusing on “how good” software does something as opposed to the functional requirements, which focus on “what” the software does.” [11].

j) "Putting it another way, NFRs constitute the justifications of design decisions and constrain the way in which the required functionality may be realized." [12].

On purpose, we left the citation to [12] as the last definition of the several presented. This definition of nonfunctional requirements is of major importance and will be commented later on in Section 4.

Since we are revisiting so many definitions, it might help to focus on the definition of four words that seem central to all of the definitions: quality, functionality, functional and nonfunctional. The definitions were selected from WordNet [13]. WordNet is a lexical database of English, where nouns, verbs, adjectives and adverbs are grouped into sets of cognitive synonyms (synsets), each expressing a distinct concept. Here we list the major definition of each of these words as they appear in Wordnet. We do this to call the attention to the term nonfunctional.

Quality: Noun -- S: (n) quality (an essential and distinguishing attribute of something or someone)

Functional: Adjective -- S: (adj) functional (designed for or capable of a particular function or use)

Nonfunctional: Adjective -- S: (adj) nonfunctional (not having or performing a function); S: (adj) malfunctioning, nonfunctional (not performing or able to perform its regular function)

Functionality: Noun -- S: (n) functionality (capable of serving a purpose well)

If we carefully examine these definitions, we notice that the word "functional" is an adjective and "quality" is both noun and adjective, whereas "functionality" is a noun. We also notice that "functional" refers to use and "functionality" refers to purpose.

We bring these definitions to bear, since if we understand the terms "functional requirements" and "non-functional requirements" out of context, they may bring up different semantics. Of course, the term "non-functional requirements" is not meant to mean requirements that are not able to perform a function, but if interpreted out of context, it may create a confusion. If the literature were more careful in choosing names, this potential confusion could have been avoided.

Notwithstanding these observations and the fact that functionality can be seen as quality as well, as also note in [6], we understand that the distinction among these two types of quality is extremely helpful and important in software engineering.

A central point for the distinction of functionality and other qualities is that, for software construction, the purpose of the software system needs to be well defined in terms of the functions that the software will perform. It may sound strange, but this distinction is not as evident in other areas of engineering, as we could see from the QFD strategy. In business and engineering, the function of an artifact or an activity mostly deals with physical entities, and is usually clear and upfront. In contrast, in software engineering whose products are conceptual entities, this is not the case. As a matter of fact, it would be odd to detail, by functions, the fact that a car has to be able to transport people, but, for a software system to be built, a software engineer has to understand what functions it should perform, usually with greater difficulty since they are not evidently visible, measurable, touchable, etc. Furthermore, software is so rapidly being applied to new application areas that it is not possible for a software engineer to build always on experiences. It is a rather well known fact that a software product may be targeting a domain not familiar to a software engineer – a problem that other types of engineers usually do not have to confront with.

The distinction between functionality and other qualities in the field of requirements engineering has an important benefit: it makes clear to software engineers that requirements are meant to deal with quality attributes and not with just one of them. As the software industry became more mature and different domains were explored by software engineers, it became clearer that it would not be enough just to deal with the description of the desired functionality, but that quality attributes should be carefully thought of early on as well.

So, in the presence of so many different definitions on NFRs, how should we proceed? We want our working definition to be as consistent with, and accommodating, other definitions. As a working definition, we start with the colloquial definition of NFRs, as in the NFR framework [1] [3], namely, any "-ilities", "-ities", along with many other things that do not necessarily end with either of them, such as performance, user-friendliness and coherence, as well as concerns on productivity, time, cost and personal happiness. In view of mathematical functions, in the form of,

$$f\colon I \rightarrow O \text{ (e.g., sum: int x int } \rightarrow \text{int)},$$

just about anything that addresses characteristics of *f, I, O* or relationships between *I* and *O* will be considered NFRs. For example, whether the summation function can easily be found on a calculator, whether the function can easily be built or modified, in a time- and cost-effective manner, whether the function returns the output fast, who can see the function, the inputs, or the output, for instance.

3 Some Classification Schemes

As seen in the previous section, various pieces of work provide for ways to distinguish among different types of quality concerns. One is the distinction between basic and extra quality [6]. Another is the distinction among concerns (sub-attributes of a quality attribute), factors (or impairments - possible properties of the system, such as policies and mechanisms built into the system, that have an impact on the concerns) and methods (means used by the system to attain the concerns) [4].

The standard ISO/IEC 9126 [14] is also noteworthy which distinguishes 4 types of quality levels: quality in use, external quality, internal quality and process quality. Based on these types, [11] provides a process oriented classification comprised of :

1) "The identification of NFR from different viewpoints and different levels of detail."
2) "The support for uncovering dependencies and conflicts between them, and to discuss and prioritize them accordingly."
3) "The documentation of NFR and the evaluation of this documentation."
4) "The support for identifying means to satisfy the NFR, to evaluate and discuss means, and to make trade-off decision accordingly. This includes cost estimation.", and
5) "The support for change and project management."

Another proposal is made in [15], using the concepts of the NFR Framework [3], on a classification of goals and softgoals, driven by the "non functional perspective". This classification provides 4 categories: functional hardgoals, nonfunctional hardgoals, functional softgoals and nonfunctional softgoals.

Another classification scheme is introduced in [16]:

- Interface requirements: describe how the system is to interface with its environment, users and other systems. E.g., user interfaces and their qualities (e.g., user-friendliness).
- Performance requirements: describe performance constraints involving
    - time/space bounds, such as workloads, response time, throughput and available storage space. E.g., "system must handle 100 transactions/second."
    - reliability involving the availability of components and integrity of information maintained and supplied to the system. E.g., "system must have less than 1hr downtime/3 months."
    - security, such as permissible information flows.
    - survivability, such as system endurance under fire, natural catastrophes.
- Operating requirements: include physical constraints (size, weight), personnel availability, skill level considerations, system accessibility for maintenance, etc.
- Lifecycle requirements: can be classified under two subcategories:
    - quality of the design: measured in terms such as maintainability, enhanceability, portability.
    - limits on development, such as development time limitations, resource availability, methodological standards, etc.
- Economic requirements: immediate and/or long-term costs
- Political requirements

Figure 1 depicts a software quality tree [17] which aims to address concerns for key types of NFRs and importantly possible correlations among them.

FURPS is an acronym representing a model for classifying software quality attributes or non-functional requirements, developed at Hewlett-Packard, and + was later added, hence FURPS+, to extend the acronym to emphasize various attributes [18]:

- Functionality - Feature set, Capabilities, Generality, Security
- Usability - Human factors, Aesthetics, Consistency, Documentation
- Reliability - Frequency/severity of failure, Recoverability, Predictability, Accuracy, Mean time to failure
- Performance - Speed, Efficiency, Resource consumption, Throughput, Response time
- Supportability - Testability, Extensibility, Adaptability, Maintainability, Compatibility, Configurability, Serviceability, Installability, Localizability, Portability

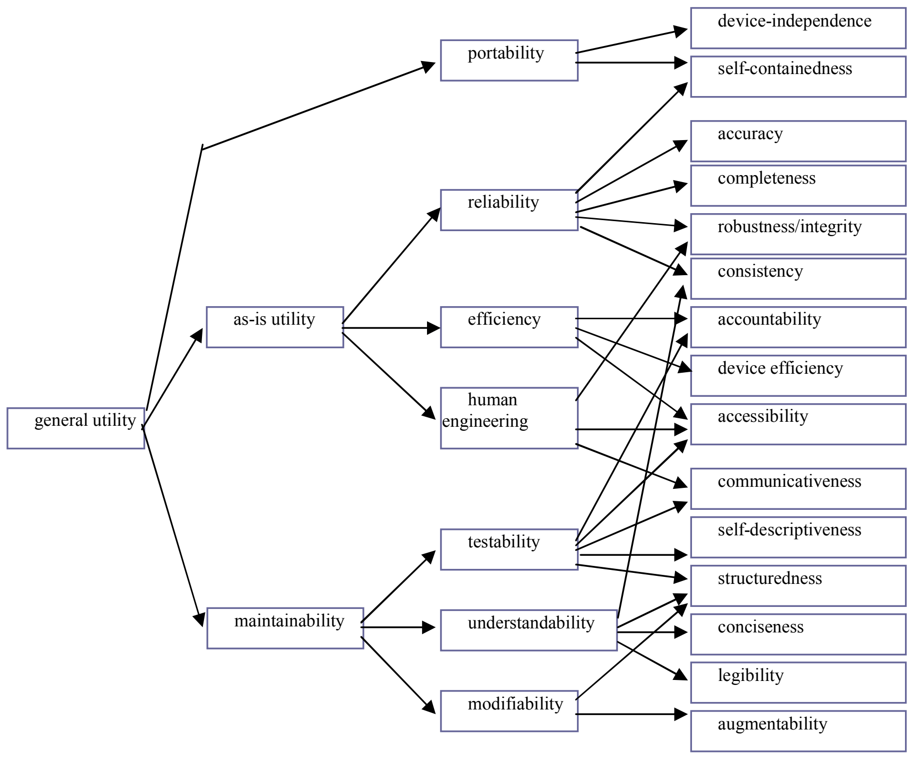


Fig. 1. Software Quality Tree [17]

However, even these well-known classification schemes are inconsistent with each other. For example, consider performance. In Roman's classification scheme, it is defined in terms 4 sub-categories, whereas it does not even appear in software quality tree, while it is shown but quite differently in FURPS+.

Another observation is that neither Roman's nor FURPS+ recognizes any potential interactions among NFRS, while software quality tree does so to a certain extent. For example, in software quality tree, portability and reliability are related to each other through a common sub-concept of self-containedness.

Similar observations can be made about other types of NFRs than performance, not only concerning the few classification schemes shown here but many other ones not shown here as well, including the one in [19].

Not surprisingly, NFRs play a critical role in the area of architectural design, and a classification scheme is provided in the context of ATAM evaluations [20], e.g., distinction of runtime qualities (availability, performance, security) from development time qualities (modifiability, integration). ATAM also addresses risk, while considering a hierarchy of architecture, process and organization.

Now, what should a software practitioner do then, when even well-known classification schemes are inconsistent with one another, not only terminologically but also categorically?

A software practitioner should be aware of some of the well known classification schemes, such as ISO 9126 - an international standard for the evaluation of software quality, and consider one or more to adopt with some tailoring. No matter what classification scheme a software practitioner might choose to adopt, the most important thing to bear in mind is that s/he should know what s/he means by an NFR term, such as performance, so that the meaning of such an NFR can be communicated with the user as well as with system/software developers so that the end product will behave as expected.

4 Representations of Non-Functional Requirements

Requirements dealing with NFRs are usually separated from the functionality requirements. A usual way to represent them is by means of requirements sentences, which are listed separately under different sections of the technical requirements section. The IEEE standard 830 - Recommended Practice for Software Requirements Specifications - is a good example, where Section 3.2 is used for specifying functional requirements, while the rest of Section 3 is used to describe different types of NFRs.

Some authors propose a structure around the requirements sentences as the one proposed by [21] that is comprised of: identification number, NFR type, use case related to it, description, rationale, originator, fit criterion, customer satisfaction, customer dissatisfaction, priority, conflicts, supporting material, and history. All of these are informal, textual information.

In the classic work with SADT [22], a requirements definition should answer three types of questions: why a system is needed (context analysis), what system features will serve to satisfy this context (system functional requirements), and how the system is to be constructed (system non-functional requirements). Although there is no explicit reference to functionality oriented requirements versus quality requirements, SADT's actigrams can be used to indirectly address some of the NFR concerns. An actigram can be associated with four types of information that interact with the activity: input, control, output and mechanism. The spirit of the control information is much related to the idea of non-functionality, since the control arrow in SADT has the purpose to constrain how the activity is performed. As such, SADT provides a way to address quality attributes or constraints.

NFRs are also commonly represented by trees, expressing the concept of NFR clustering or decomposition [4] and also by lists as well.

Some authors have used NFRs in conjunction with a more structured requirements representation notation, e.g., the combination of NFRs with use cases or misuse cases (for example, see [23], [24],[25], [26] and [27]).

NFRs are also represented as restrictions over different parts of a scenario, along with time and location as the contextual information in the scenario description [28]. Here, the representation of a scenario is comprised of: title, goal, context, resources, actors, episodes, exceptions and the attribute constraint, which applies to context, resources and episodes. The entity context is further divided into geographical location, time and pre-condition.

Among many proposals, however, the goal oriented approaches, as in [2] [3], were the first to treat NFRs in more depth.

In KAOS [2], which perhaps pioneered in promoting goal-oriented requirements engineering at least from a functional goal perspective, goals are: "… modelled by intrinsic features such as their type and attributes, and by their links to other goals and to other elements of a requirements model." KAOS addresses both functional goals and non-functional goals, which are formalized in terms of operators, such as Achieve, Maintain and Avoid, and by activities based on temporal logic augmented with some special temporal operators. For instance, KAOS offers special operators for concepts, such as "sometime in the future" and "always in the future unless". Thanks to their formal nature, representations in KAOS are amenable to automatic verification and reasoning. In KAOS, goals are characterized as a set of high level constraints over states. For instance, an informal goal: {{Goal Maintain [DoorsClosedWhileMoving]}} [2] can be expressed by the following formula:

$$\{\{\forall\ tr\text{: } Train,\ loc,\ loc'\text{: } Location\ At\ (tr, loc) \wedge o\ At\ (tr, loc') \wedge loc <> loc' \Rightarrow tr.Doors = \text{'closed'} \wedge o\ (tr.Doors = \text{'closed'})\}\}.$$

The above formula, where "o" is a temporal operator denoting next state, means that while a train moves from one location to another location, its doors much be closed during the move.

Although the KAOS' representation language does not differentiate between functional and non-functional goals, the KAOS graphical AND/OR graph makes the differentiation. Goals that can be assigned to individual agents need no further decomposition and can be "operationalized", that is, they can be described in terms of constraints.

Not unlike KAOS, the NFR Framework also promotes goal orientation, but with the main emphasis on NFRs. In the NFR Framework, non-functional requirements are treated as softgoals, i.e., goals that need to be addressed not absolutely but in a good-enough sense. This is in recognition of the difficulties that are associated with both the problem and the corresponding solution. Concerning the problem statement, it is often times extremely difficult, if not impossible, to define an NFR term completely unambiguously without using any other NFR term, which in turn will have to be defined. Concerning the solution, it is also often times extremely difficult to explore a complete list of possible solutions and choose the best, or optimal, solution, due to various resource limitations such as the time, manpower and money available for such an exploration.

Reflecting the sense of "good enough", the NFR Framework introduces the notion of satisficing, and, with this notion, a softgoal is said to satisfice (instead of satisfy) another softgoal. The NFR Framework offers several different types of contributions whereby a softgoal satisfices, or denies, another softgoal - MAKE, HELP, HURT and BREAK are the prominent ones, as well as AND and OR (these also incorporate the notion of "good enough" rather than "absolute satisfaction"). MAKE and HELP are used to represent a softgoal positively satisficing another, while BREAK and HURT to represent a softgoal negatively satisficing (or denying) another . While MAKE and

BREAK respectively reflect our level of confidence in one softgoal fully satisficing or denying another, HELP and HURT respectively reflect our level of confidence in one softgoal partially satisficing or denying another.

In the NFR Framework, each softgoal or contribution is associated with a label, indicating the degree to which it is satisficed or denied. A label propogation procedure is offered in the NFR Framework in order to determine the effect of various design decisions, regardless of whether they are system-level or software-level. In addition to a label, each softgoal or contribution can also be associated with a criticality value, such as extremely critical, critical, and non-critical.

When using the NFR Framework, NFRs are posted as softgoals to be addressed or achieved, and an iterative and interleaving process is applied in order to satisfice them, through AND/OR decompositions, operationalizations and argumentations. Throughout the process, a visual representation, SIG (softgoal interdependency graph), is created and maintained which keeps track of softgoals and their interdependencies, along with the impact of various decisions through labels. In this sense, a SIG shows how various (design) decisions are rationalized.

In order to alleviate the difficulties associated with the search for knowledge for dealing with NFRs , the NFR Framework offers methods for capturing knowledge of ways to satisfice NFRs and correlation rules for knowledge of the side effects that such methods induce.

As with goals in KAOS, softgoals in the NFR Framework are associated with, and ultimately achieved, by the actions of agents – human, hardware or software. This is consistent with the spirit of the reference model [29], in which (functional) requirements are satisfied through the collaboration between the software system behavior and environment phenomena that are caused by agents in the environment, although here we are also concerned with softgoals that (functional) requirements are intended to help achieve together with environment phenomena. Note that requirements are part of the solution to some problem in a piece of reality, and the notion of softgoals can be used to represent anything non-functional, be it about the problem domain or the solution.

The i* family: i* [30], Tropos [31] and GRL(Goal Requirements Language) [32] inherited the concept of softgoal from the NFR Framework, aiming at dealing with softgoals, or non-functionality related attributes, as a first class modeling concept.

As mentioned in Section 2, the last definition [12] presented was somewhat different from the rest: an NFR is described as a justification of a design decision and as a constraint on the way in which a required functionality may be realized. This is exactly why the proper identification and representation of NFRs play a key role in software engineering, since software engineering is said to be all about decision-making.

As several authors have pointed out, NFRs do need to be transformed through some means, methods or operations. This is also why a goal-oriented representation is so well suited for NFRs - they are initially expressed in general terms as more abstract requirements, but then gradually are further refined into more concrete terms and details.

When NFRs eventually are operationalized, in terms of software operations, entities or constraints, they become the justification for why such operationalizations exist in the software system, i.e., to serve the quality attributes specified as NFR softgoals. If the software engineers are careful enough to maintain the history of the software construction, they will then be able to explain and justify why such operationalizations exist. It goes without saying that this argument is also valid for functionality as well, but the key point here is that the choice on a specific operation to reify a quality concern affects how the overall functionality is achieved. Put differently, different sorts of design decisions are made throughout a software development process, and NFRs act as the criteria for such design decisions (See [33] for a discussion about these design decisions in the context of use cases). As argued in [12], an NFR is not just an after-the-fact justification, but constrains how functionality is realized.

So, if the software engineer uses quality attributes up front during the software development process, there will be a network of explanations, bounded by the usage of rationality, linking the results of decisions with the quality attributes. Work on design rationale [34], drawing on earlier work on the Ibis idea [35], focuses on the justification of decisions, but without taking into consideration pre-existing factors that lead to the decision. Work on design rationale can benefit from better ways of dealing with the dynamic and contextual nature of software design and with the limitations of working with the myriad of possible alternatives and their justifications [36].

Integration of functionality and other qualities is said to be essential. Although no one would dispute that truism, few have proposed or advocated a process that really intertwines these two classes of requirements. Goal-oriented methods, such as the NFR Framework, KAOS and the i* family, are the few exceptions that make the consideration of non-functionality as a first class concept, being intertwined with the functionality, as they are reified. However, it shouldn't come as a surprise that each approach has its own particular ways of doing this interweaving of functionality and quality attributes, with several distinct variations and without necessarily keeping the development history.

So what? In a nutshell, the point we are trying to make is twofold: not only non-functional requirements need to be stated up front, but they can help the software engineer make design decisions, while also justifying such decisions. However, in order to take this into consideration, it is necessary that quality attributes not be considered just as a separate set of requirements, but with the consideration of the functionality throughout the development process.

The NFR Framework and the i* family have an intrinsic characteristic that sets them apart from other NFR methods - the reliance on a qualitative approach towards NFRs or softgoals. In the heart of the NFR Framework lies the concept that softgoals are idealizations and, as such, without all its defining properties necessarily established. This characteristic is similar to the notion of "bounded rationality" put forward by Herbert Simon [37] when explaining his understanding of the process designers use to make a decision given incomplete information. This qualitative characteristic is built on the ideas of contribution and correlation among softgoals as explained earlier. By means of contributions, a softgoal may be decomposed up to the level of operationalization, and, by correlations, different softgoals may interfere among themselves. A network of such contributions and correlations makes it possible to carry out different sorts of qualitative reasoning. The semantics of such a network is given by relationships over three dimensions a) decompositions over an AND/OR tree, b) contributions among sub-trees and c) correlations among different softgoals, all leading to the formation of a softgoal interdependency graph (SIG).

While the NFR Framework's focus is on NFRs, as described at the end of Section 2, NFRs exist in relation to functional things. UML is a functionally-oriented Object-Oriented analysis and design language, and one of the first works to detail how the NFR Framework could help attain better UML models is described in [38], which presents a process for linking NFR graphs with UML models. The central idea is to qualitatively realize softgoals in the UML models. The realization for UML classes, for instance, was based on introducing attributes to classes, methods, or constraints over the attributes.

In i* [30], the links among softgoals and operations (tasks or resources) are more explicit, since modeling is carried out simultaneously in the context of the Strategic Rationale (SR) diagram. In a SR diagram, means-ends relationships (i* specialization operator) can show how choices (tasks) are related to different softgoals, while also showing the pros and cons of each selection.

The NFR Framework presented in this Section accommodates any classification scheme that was discussed in Section 3, through AND/OR decompositions, and goes beyond by offering those concepts of operationalizations and argumentations, together with positive/negative correlations. In terms of these concepts, the NFR Framework helps rationalize design decisions – both system-level and software-level. For representation of NFRs, it recognizes NFRs as softgoals and relatioships between them as partial/full positive/negative contributions.

5 Future Directions

There have been several pieces of work that further explored the concepts of softgoals, while shaping some scenario for future directions. We classify them in six areas: software variability, requirements analysis, requirements elicitation, requirements reusability, requirements traceability and aspect-oriented development. Each of these areas has explored particular aspects of the idea of a SIG (Softgoal Interdependency Graph).

When dealing with software product lines, the idea of variability is critical, since, at some parts of a product line, architecture variation points will exist to enable the production of different alternatives necessary to compose different products out of a single common architecture. The works of [39] [40] [41] explored the fact that the alternatives are intrinsic in SIG models, since they are AND/OR graphs. Using a goal-oriented approach to product lines brings a seamless way of producing product line architectures, since features are not only identified, but also justified, in terms of softgoals.

One of the major advantages of the qualitative approach of a SIG is that it facilitates analysis. The very idea that there can be different types of relationships among softgoals and between softgoals and operationalizations in a SIG brings the opportunity to conduct analysis, by propagating the impact of decisions along the correlation contribution relationships. Using the concept of label propagation over a SIG graph, it is possible to evaluate how a given operationalization of an NFR will impact the whole graph. The original process was devised in the NFR Framework [3] and variations followed [42] [43]. Using the idea of label propagation, different types of analysis could be performed early on before committing to an architecture or to code, as seen in the exploration of security concerns [44], in visually choosing operationalizations [45], and in casting an i* model analysis as a SAT problem [46].

Eliciting requirements requires use of different sets of techniques, but most of them are centered on discovering functionality only. Work on goal elicitation needs to consider both functionality and other qualities. Work in this regard includes a proposal on an elicitation scheme that departs from an extended lexicon [47], repertory grid techniques to help the clarification of naming during the elicitation process [48], and Personal Construct Theory to elicit contribution among softgoal [49].

The NFR Framework [3] has identified that NFR catalogues, composed of SIG graphs, were an important aspect of building software using the softgoal concept. Later work [50] explored the ideas further on stressing the aspect of reusability, while proposing a method for maintaining a softgoal organization aimed for reuse. The idea of a goal centric traceability based on softgoal graphs is explored in [51] [52], in which the softgoal network is used as a baseline for explaining changes over software evolution.

The relationship among the quality attributes and aspect-oriented development has been explored in [53] and [54]. Later on, others have identified the important role the NFR Framework - in particular, softgoals - plays in dealing with early aspects [27], [55] [56] [57] [58] [59] (See [60] for a survey on the topic).

Although these recent results helped further understanding of the general concept of quality requirements and opened new paths for future research, other issues need further advance as well, such as the integration of NFRs into other requirements models, such as the reference model [29] and the four variable model [61] which have had significant influences in the area of requirements engineering. Although these models address performance or accuracy concerns, they are essentially functional models and without the notion of goals. As briefly mentioned in Section 4, for example, the reference model states that (functional) requirements are satisfied through the collaboration between the functional behavior of the software system and the (functional) phenomena in the environment. KAOS [2] goes beyond these functional models and introduces general types of softgoals for the overall system, while addressing performance, accuracy and security concerns for the software system.

We also agree with [11] on the need for further empirical research on the use of NFRs during requirements engineering and on the usage of ethnographical studies on how software teams deal with quality issues as requirements. We understand that this research should be conducted with real projects, both in lab situations as well as on industry projects, to improve our knowledge on dealing with quality issues early on.

Acknowledgements. We appreciate the comments from Barbara Paech on an earlier draft, which significantly helped us improve the paper in a more understandable manner.

References

1. Mylopoulos, J., Chung, L., Nixon, B.: Representing and Using Nonfunctional Requirements: A Process-Oriented Approach. IEEE Trans. Softw. Eng. 18(6), 483–497 (1992), http://dx.doi.org/10.1109/32.142871
2. van Lamsweerde, A.: Goal-Oriented Requirements Engineering: A Guided Tour. In: Proceedings of the 5th IEEE international Symposium on Requirements Engineering, August 27-31, 2001, p. 249. IEEE Computer Society, Washington (2001)

3. Chung, L., Nixon, B.A., Yu, E., Mylopoulos, J.: Non-Functional Requirements in Software Engineering. International Series in Software Engineering, vol. 5, p. 476. Springer, Heidelberg (1999)
4. Barbacci, M., Longstaff, T.H., Klein, M.H., Weinstock, C.B.: Quality Attributes, Technical Report CMU/SEI-95-TR-021, ESC-TR-95-021 (December 1995)
5. IEEE Standard 1061-1992 Standard for a Software Quality Metrics Methodology. Institute of Electrical and Electronics Engineers, New York (1992)
6. Freeman, P.A.: Software Perspectives: The System is the Message. Addison-Wesley, Reading (1987)
7. QFD Institute, Quality Function Deployment, `http://www.qfdi.org/`
8. Hauser Jr., Clausing, D.: The house of quality. Harvard Business Review 66(3), 63–73 (1988)
9. Yeh, R.T., Zave, P., Conn, A.P., Cole, G.E.: Software Requirements Analysis — New Directions and Perspectives. In: Vick, C.R., Ramamoorthy, C.V. (eds.) Handbook of Software Engineering, Van Nostrand Reinhold Co. (1984)
10. Glinz, M.: On Non-Functional Requirements. In: 15th IEEE International Requirements Engineering Conference (RE 2007), pp. 21–26 (2007)
11. Paech, B., Kerkow, D.: Non-Functional Requirements Engineering - Quality is Essential. In: 10th Anniversary International Workshop on Requirements Engineering: Foundation for Software Quality, REFSQ 2004 (2004), `http://www.sse.uni-essen.de/refsq/downloads/toc-refsq04.pdf`
12. Landes, D., Studer, R.: The Treatment of Non-Functional Requirements in MIKE. In: Botella, P., Schäfer, W. (eds.) ESEC 1995. LNCS, vol. 989, pp. 294–306. Springer, Heidelberg (1995)
13. A lexical database of English, `http://wordnet.princeton.edu/`
14. ISO/IEC 9126-1:2001(E): Software Engineering - Product Quality - Part 1: Quality Model (2001)
15. Jureta, I.J., Faulkner, S., Schobbens, P.-Y.: A more expressive softgoal conceptualization for quality requirements analysis. In: Embley, D.W., Olivé, A., Ram, S. (eds.) ER 2006. LNCS, vol. 4215, pp. 281–295. Springer, Heidelberg (2006)
16. Roman, G.-C.: A Taxonomy of Current Issues in Requirements Engineering. IEEE Computer, 14–21 (April 1985)
17. Boehm, B.W., Brown, J.R., Kaspar, H., Lipow, M., MacLeod, G.J., Merritt, M.J.: Characteristics of Software Quality. North-Holland, Amsterdam (1978)
18. Grady, R., Caswell, D.: Software Metrics: Establishing a Company-wide Program. Prentice-Hall, Englewood Cliffs (1987)
19. Bowen, T.P., Wigle, G.B., Tsai, J.T.: Specification of Software Quality Attributes, Report RADC-TR-85-37, vol. I (Introduction), vol. II (Software Quality Specification Guidebook), vol. III (Software Quality Evaluation Guidebook), Rome Air Development Center, Griffiss Air Force Base, NY (February 1985)
20. Bass, L., Nord, R., Wood, W., Zubrow, D.: Risk Themes Discovered Through Architecture Evaluations, Technical Report CMU/SEI-2006-TR-012, ESC-TR-2006-012 (2006)
21. Robertson, S., Robertson, J.: The Volere requirements process, Mastering the Requirements Process. Addison-Wesley, London (1999)
22. Ross, D.T.: Structured Analysis (SA): A Language for Communicating Ideas. IEEE Trans. Softw. Eng. 3(1), 16–34 (1977), `http://dx.doi.org/10.1109/TSE.1977.229900`
23. Chung, L., Supakkul, S.: Representing nFRs and fRs: A goal-oriented and use case driven approach. In: Dosch, W., Lee, R.Y., Wu, C. (eds.) SERA 2004. LNCS, vol. 3647, pp. 29–41. Springer, Heidelberg (2006)
24. Herrmann, A., Paech, B.: MOQARE: misuse-oriented quality requirements engineering. Requir. Eng. 13(1), 73–86 (2008)

25. Cysneiros, L.M., do Prado Leite, J.C.: Using UML to reflect non-functional requirements. In: Stewart, D.A., Johnson, J.H. (eds.) Proceedings of the 2001 Conference of the Centre For Advanced Studies on Collaborative Research. IBM Centre for Advanced Studies Conference, vol. 2. IBM Press (2001)
26. Alexander, I.: Misuse cases help to elicit non-functional requirements. Computing & Control Engineering Journal 14(1), 40–45 (2003)
27. de Sousa, T.G.M.C., Castro, J.F.B.: Towards a Goal-Oriented Requirements Methodology Based on the Separation of Concerns Principle. In: Anais do WER 2003 - Workshop em Engenharia de Requisitos, Piracicaba-SP, Brasil, November 27-28, 2003, pp. 223–239 (2003), http://wer.inf.puc-rio.br/WERpapers/artigos/artigos_WER03/georgia_souza.pdf
28. Leite, J.C., Hadad, G., Doorn, J., Kaplan, G.: A Scenario Construction Process. Requirements Engineering Journal 5(1), 38–61 (2000)
29. Gunter, C., Gunter, E., Jackson, M., Zave, P.: A Reference Model for Requirements and Specifcations. IEEE Software, 37–43 (2000)
30. Yu, E.S.K.: Towards modelling and reasoning support for early-phase requirements engineering. In: Proceedings of the Third IEEE International Symposium on Requirements Engineering, pp. 226–235 (1997)
31. Castro, J., Kolp, M., Mylopoulos, J.: Towards requirements-driven information systems engineering: the Tropos project. Information Systems 27(6), 365–389 (2002)
32. Amyot, D., Mussbacher, G.: URN: Towards a new standard for the visual description of requirements. In: Sherratt, E. (ed.) SAM 2002. LNCS, vol. 2599, pp. 21–37. Springer, Heidelberg (2003)
33. Dutoit, A.H., Paech, B.: Rationale-based use case specification. Requirements engineering 7(1), 1–3 (2002)
34. Potts, C., Bruns, G.: Recording the reasons for design decisions. In: Proceedings of the 10th international Conference on Software Engineering. International Conference on Software Engineering, Singapore, April 11-15, 1988, pp. 418–427. IEEE Computer Society Press, Los Alamitos (1988)
35. Kunz, W., Rittel, H.W.J.: Issues as Elements of Information Systems, Working Paper No. 131 (July 1970); Studiengruppe für Systemforschung, Heidelberg, Germany (reprinted, May 1979)
36. Dutoit, A.H., McCall, R., Mistrík, I., Paech, B. (eds.): Rationale Management in Software Engineering. Springer, Heidelberg (2006)
37. Simon, H.A.: The Sciences of the Artificial, 3rd edn. The MIT Press, Cambridge, MA (1977)
38. Cysneiros, L.M., Leite, J.C.: Nonfunctional Requirements: From Elicitation to Conceptual Models. IEEE Trans. Softw. Eng. 30(5), 328–350 (2004), http://dx.doi.org/10.1109/TSE.2004.10
39. Liaskos, S., Lapouchnian, A., Yu, Y., Yu, E.S.K., Mylopoulos, J.: On Goal-based Variability Acquisition and Analysis. In: RE 2006, pp. 76–85 (2006)
40. Yu, Y., Lapouchnian, A., Liaskos, S., Mylopoulos, J., Leite, J.C.: From goals to high-variability software design. In: An, A., Matwin, S., Raś, Z.W., Ślęzak, D. (eds.) Foundations of Intelligent Systems. LNCS, vol. 4994, pp. 1–16. Springer, Heidelberg (2008)
41. González-Baixauli, B., Laguna, M.A., Leite, J.C.: Using Goal-Models to Analyze Variability. In: First International Workshop on Variability Modelling of Software-Intensive Systems, VaMoS 2007, Proceedings, Limerick, Ireland, January 16-18, 2007, pp. 101–107, Lero Technical Report 2007-01 2007 (2007)
42. Giorgini, P., Mylopoulos, J., Nicchiarelli, E., Sebastiani, R.: Reasoning with Goal Models. In: Spaccapietra, S., March, S.T., Kambayashi, Y. (eds.) ER 2002. LNCS, vol. 2503, pp. 167–181. Springer, Heidelberg (2002)

43. Kaiya, H., Horai, H., Saeki, M.: AGORA: Attributed Goal-Oriented Requirements Analysis Method. In: Proceedings of the 10th Anniversary IEEE Joint international Conference on Requirements Engineering, September 09-13, 2002, pp. 13–22. IEEE Computer Society, Washington (2002)
44. Liu, L., Yu, E., Mylopoulos, J.: Security and Privacy Requirements Analysis within a Social Setting. In: Proceedings of the 11th IEEE international Conference on Requirements Engineering, September 08-12, 2003, IEEE Computer Society, Washington (2003)
45. Gonzalez-Baixauli, B., Leite, J.C., Mylopoulos, J.: Visual Variability Analysis for Goal Models. In: Proceedings of the Requirements Engineering Conference, 12th IEEE international, September 06-10, 2004, pp. 198–207. IEEE Computer Society, Washington (2004), http://dx.doi.org/10.1109/RE.2004.56
46. Horkoff, J., Yu, E.S.K.: Qualitative, Interactive, Backward Analysis of i* Models. In: iStar 2008, pp. 43–46 (2008)
47. Oliveira, A.P.A., Leite, J.C., Cysneiros, L.M.: AGFL - Agent Goals from Lexicon - Eliciting Multi-Agent Systems Intentionality. In: iStar 2008, pp. 29–32 (2008)
48. Niu, N., Easterbrook, S.M.: Managing Terminological Interference in Goal Models with Repertory Grid. In: RE 2006, pp. 296–299 (2006)
49. González-Baixauli, B., Leite, J.C., Laguna, M.A.: Eliciting Non-Functional Requirements Interactions Using the Personal Construct Theory. In: RE 2006, pp. 340–341 (2006)
50. Cysneiros, L.M., Werneck, V., Kushniruk, A.: Reusable Knowledge for Satisficing Usability Requirements. In: RE 2005, pp. 463–464 (2005)
51. Cleland-Huang, J., Settimi, R., BenKhadra, O., Berezhanskaya, E., Christina, S.: Goal-centric traceability for managing non-functional requirements. In: Proceedings of the 27th international Conference on Software Engineering, ICSE 2005, St. Louis, MO, USA, May 15-21, 2005, pp. 362–371. ACM, New York (2005), http://doi.acm.org/10.1145/1062455.1062525
52. Cleland-Huang, J., Marrero, W., Berenbach, B.: Goal-Centric Traceability: Using Virtual Plumblines to Maintain Critical Systemic Qualities. IEEE Trans. Softw. Eng. 34(5), 685–699 (2008), http://dx.doi.org/10.1109/TSE.2008.45
53. Grundy, J.C.: Aspect-Oriented Requirements Engineering for Component-Based Software Systems. In: Proceedings of the 4th IEEE international Symposium on Requirements Engineering, RE, June 07-11, 1999, pp. 84–91. IEEE Computer Society, Washington (1999)
54. Moreira, A., Araújo, J., Brito, I.: Crosscutting quality attributes for requirements engineering. In: Proceedings of the 14th international Conference on Software Engineering and Knowledge Engineering, SEKE 2002, Ischia, Italy, July 15-19, 2002, vol. 27, pp. 167–174. ACM, New York (2002), http://doi.acm.org/10.1145/568760.568790
55. Yu, Y., Leite, J.C., Mylopoulos, J.: From Goals to Aspects: Discovering Aspects from Requirements Goal Models. In: 12th IEEE international Proceedings of the Requirements Engineering Conference, September 06-10, 2004, pp. 38–47. IEEE Computer Society, Washington (2004), http://dx.doi.org/10.1109/RE.2004.23
56. Brito, I., Moreira, A.: Integrating the NFR framework in a RE model. In: EA-AOSD 2004: Workshop on Early Aspects: Aspect-Oriented Requirements Engineering and Architecture Design, held in conjunction with the 3rd International Conference on Aspect-Oriented Software Development, Lancaster, UK, March 22-26 (2004), http://trese.cs.utwente.nl/workshops/early-aspects-2004/Papers/BritoMoreira.pdf
57. Alencar, F., Silva, C., Moreira, A., Araújo, J., Castro, J.: Identifying Candidate Aspects with I-star Approach. In: Early Aspects 2006: Traceability of Aspects in the Early Life Cycle, pp. 4–10 (2006)

58. de Padua Albuquerque Oliveira, A., Cysneiros, L.M., do Prado Leite, J.C., Figueiredo, E.M., Lucena, C.J.: Integrating scenarios, i*, and AspectT in the context of multi-agent systems. In: Proceedings of the 2006 Conference of the Center For Advanced Studies on Collaborative Research, CASCON 2006, Toronto, Ontario, Canada, October 16-19, 2006, p. 16. ACM, New York (2006), `http://doi.acm.org/10.1145/1188966.1188988`
59. da Silva, L.F., Leite, J.C.: Generating Requirements Views: A Transformation-Driven Approach. ECEASST 3 (2006)
60. Yu, Y., Niu, N., González-Baixauli, B., Mylopoulos, J., Easterbrook, S., Leite, J.C.: Requirements Engineering and Aspects. In: Design Requirements Engineering: A Ten-Year Perspective. Lecture Notes in Business Information Processing, pp. 432–452. Springer, Heidelberg (2009)
61. Parnas, D.L., Madey, J.: Functional Documentation for Computer Systems. Science of Computer Programming 25(1), 41–61 (1995)

Reasoning About Alternative Requirements Options[*]

Axel van Lamsweerde

Département d'Ingénierie Informatique, Université catholique de Louvain,
Place Sainte Barbe 2, B-1348 Louvain-la-Neuve (Belgium)
avl@info.ucl.ac.be

Abstract. This paper elaborates on some of the fundamental contributions made by John Mylopoulos in the area of Requirements Engineering. We specifically focus on the use of goal models and their soft goals for reasoning about alternative options arising in the requirements engineering process. A personal account of John's qualitative reasoning technique for comparing alternatives is provided first. A quantitative but lightweight technique for evaluating alternative options is then presented. This technique builds on mechanisms introduced by the qualitative scheme while overcoming some problems raised by it. A meeting scheduling system is used as a running example to illustrate the main ideas.

1 Introduction

Poor requirements were recurrently recognized to be the major cause of software problems such as cost overruns, delivery delays, failure to meet expectations, or degradations in the environment controlled by the software. The early awareness of the so-called requirements problem [2], [3], [6] raised preliminary efforts to develop modeling languages for requirements definition and analysis [1], [40], [18]. With the increasing complexity of software-intensive systems, the research challenges raised by the requirements problem were so significant that an active community emerged in the nineties with dedicated conferences, workshops, working groups, networks, and journals. The term "*requirements engineering*" (RE) was introduced to refer to the process of eliciting, evaluating, specifying, consolidating, and changing the objectives, functionalities, qualities, and constraints to be achieved by a software-intensive system.

John Mylopoulos was involved in requirements engineering research since the early days. His ICSE'82 paper brought early RE research efforts significantly further [16]. The *SADT* graphical language [40] allowed analysts to model two dual system views, the data view and the operation view, together with rudimentary forms of events, triggering operations, and performing agents. As in *Structured Analysis* [13], stepwise model refinement was supported. The *RML* language introduced in [16] provided richer structuring mechanisms such as generalization, aggregation and classification. In that sense it was a precursor to object-oriented analysis techniques. These structuring mechanisms were applicable to three kinds of conceptual units:

[*] Sections 4 and 5 of this paper are slight adaptations of Sections 16.3.1 and 16.3.2 in Chapter 16 of A. van Lamsweerde, *Requirements Engineering: From System Goals to UML Models to Software Specifications*, Wiley, 2009.

entities, operations, and constraints. Constraints were expressed in a formal assertion language providing, in particular, built-in constructs for temporal referencing. *RML* was also the first requirements modeling language to have a formal semantics, defined in terms of mappings to first-order predicate logic [17]. The *RML* breakthrough was made possible, I believe, thanks to John's unique position at the intersection of three different areas: database modeling [41], [33], knowledge representation [5], and formal specification [4].

In the early nineties, John's group and mine converged on two observations (without any interaction at that time):

- Models for RE needed richer and higher-level abstractions than those provided by design modeling languages and software specification techniques. (It took more than a decade for this observation to gain wide acceptance [32], [36], [22], [24]).
- Some well-established AI techniques were relevant to the RE challenges we wanted to address, in particular in the area of general problem solving, knowledge representation, and knowledge acquisition [21], [34].

Goal-Oriented Requirements Engineering (GORE) emerged from these premises. Two GORE frameworks appeared independently: *KAOS* [10], [11] and *NFR/i** [34], [43]. While both frameworks addressed common targets such as modeling goals and their responsible agents, there were differences and complementarities in focus.

- By and large, the emphasis in NFR/i* was more on qualitative reasoning about soft goals for: analysis of goal contributions [8]; evaluation of alternative goal refinements [35]; reasoning about dependencies among organizational agents [43] and vulnerabilities resulting from such dependencies [31]; customization of user-software interactions [19]; and transition to agent-oriented programming [7].
- In KAOS, the emphasis was more on semi-formal and formal reasoning about behavioral goals for: derivation of goal refinements [12], goal operationalizations [29], and goal assignments [28]; goal-based risk analysis [27]; conflict management [26]; and behavior model synthesis from goals and scenarios [9].

KAOS was more oriented towards goal satisfaction whereas NFR/I* was more oriented towards goal satisficing. Beyond our complementary approaches, there were parallel efforts towards formal goal-based model checking and animation [15], [42], [38] and security analysis [31], [23].

In this overall setting, this paper focuses on what appears to me among the most important contributions of John's group to RE, namely,

- the introduction and use of soft goals as criteria for evaluating alternative options arising throughout the RE process;
- the exploitation of goal models for evaluating such options through qualitative reasoning schemes.

This choice of focus is inevitably biased by my research interests and by the ways John's work has influenced my own efforts. The purpose of the paper is to provide a brief personal account of John's work in this area and discuss some continuation aimed at addressing various issues raised by it.

Section 2 reviews the various types of alternative options we can find during requirements elicitation and evaluation. Section 3 briefly discusses goal models together with the role played by soft goals in them. Section 4 outlines the qualitative reasoning technique in [34], illustrates its use on a meeting scheduling system, and discusses some problems we may experience with it. Section 5 presents a lightweight quantitative technique aimed at addressing these problems while sharing the same underlying principles.

2 Alternative Options in Requirements Engineering

In any software project, we need to discover, understand, formulate, analyze and agree on *what* problem should be solved, *why* such problem needs to be solved, and *who* should be involved in the responsibility of solving that problem. The problem arises within some broader context. It is in general rooted in a complex organizational, technical, or physical system. This *system* is made of components such as organizational units, people playing specific roles, devices such as measurement instruments, sensors and actuators, and pre-existing software. The aim of the project is to improve this system by building a software solution to the problem and plugging it into the system. We therefore need to consider two versions of the same system:

- the *system-as-is* is the system as it exists before the software is built into it,
- the *system-to-be* is the system as it should be when the software will be built and operate in it.

Our job as requirements engineers is to explore the desired effects of the software-to-be on its surrounding environment together with the assumptions we need to make about this environment. While doing so, we have to make decisions among alternative options arising at multiple places [25], in particular:

- When we have elicited an objective of the system-to-be and want to decompose it into sub-objectives; different decompositions might be envisioned, and we need to select a "best" one.
- When we have identified some likely and critical risk; different countermeasures might be envisioned, and we need to select a "best" one.
- When we have detected a conflict between objectives or requirements and want to resolve it; different resolutions might be envisioned, and we need to select a "best" one.
- When we operationalize a system objective through some combination of functional features, constraints, and assumptions; different combinations might be envisioned, and we need to select a "best" one.
- When we consider alternative assignments of responsibilities among components of the system-to-be –in particular, alternative software-environment boundaries where more or less functionality is automated; "best" alternatives must eventually be selected.

All such situations involve *system* design decisions (not to be confused with software design decisions). Once such decisions have been made, we need to recursively elicit, evaluate, document, and consolidate new requirements and assumptions based on them. Different decisions result in different system proposals which, in turn, will result in different software architectures.

Consider a meeting scheduling system, for example. The objective of knowing the constraints of invited participants might be decomposed into a sub-objective of knowing these constraints through email requests or, alternatively, a sub-objective of knowing them through access to their electronic agenda. A system based on e-mail communication for getting constraints will be different at places from one based on e-agendas. Likewise, different system proposals will result from an alternative where meeting initiators are taking responsibility for resolving date conflicts and a more automated alternative where the software-to-be is responsible for this.

3 Goal Models and the Role of Soft Goals

The system-to-be is intended to meet a number of objectives. These are highlighted as first-class citizens in a goal model where they are interrelated through positive/negative contribution links. A *goal* is a prescriptive statement of intent the system should satisfy through cooperation of its agents. An *agent* is an active system component having to play some role in goal satisfaction through adequate control of system items [25].

Goal satisfaction may involve a variety of system agents defining the *system scope*. The finer-grained a goal is, the fewer agents are required to satisfy it. A *requirement* is a goal under responsibility of a single agent of the software-to-be. An *expectation* is a goal under responsibility of a single agent in the environment of the software-to-be. Expectations cannot be enforced by the software-to-be; they form one kind of assumption we need to make for the system to satisfy its goals.

To be under the sole responsibility of an agent, a goal must be *realizable* by this agent [28]. This means roughly that the agent must be able to control the state variables constrained by the goal specification and to monitor the state variables to be evaluated in this specification.

While reasoning about goal satisfaction in the RE process, we often need to use *domain properties* [25]. These are descriptive statements about the system unlike goals; the latter are prescriptive. Domain properties are expected to hold invariably regardless of how the system will behave [20], [37]. The distinction between descriptive and prescriptive statements is important. Goals may need to be refined into subgoals, negotiated with stakeholders, assigned to agents responsible for them, weakened in case of conflict, or strengthened or discarded in case of unacceptable exposure to risks. Unlike prescriptive statements, domain properties are not subject to such decisions in the RE process.

A goal is either a behavioral goal or a soft goal. A *behavioral goal* prescribes intended system behaviors declaratively. It implicitly defines a maximal set of admissible behaviors [11]. Behavioral goals can be *Achieve* or *Maintain/Avoid* goals. An *Achieve goal* prescribes some *TargetCondition* to be established sooner or later when some current condition holds. A *Maintain goal* prescribes some *GoodCondition* to be maintained. (Similarly, an *Avoid goal* prescribes some *BadCondition* to be avoided.)

Unlike behavioral goals, a *soft goal* cannot be established in a clear-cut sense [34]. In a meeting scheduling system, for example, we cannot say in a strict sense whether a specic system behavior satisfies the goal of reducing the meeting initiator's load or not. We may however say that one system behavior reduces the initiator's load further than another. Said in more general terms, the phrase "goal satisfaction" should not be taken in a strict sense for a soft goal as we cannot observe that the goal is satisfied by some behaviors and not satisfied by others. The phrase "*goal satisficing*" is sometimes used instead; the degree of satisfaction of a soft goal may be higher in one alternative than in another.

Behavioral goals are therefore used for deriving system operations to satisfy them [11], [29] whereas soft goals are used for comparing alternative options to select best ones [34], [8]. We come back to this in Sections 4 and 5.

The behavioral/soft goal typology should not be confused with a goal categorization into functional goals, underlying system services, and non-functional goals, prescribing qualiy of service. For example, a confidentiality goal *Avoid[SensitiveInformationDisclosed]* is traditionally considered as non-functional; it is not a soft goal though as we can always determine whether or not this goal is satisfied in a clear-cut sense.

A *goal model* is basically an annotated AND/OR graph showing how higher-level goals are satisfied by lower-level ones (goal *refinement*) and, conversely, how lower-level goals contribute to the satisfaction of higher-level ones (goal *abstraction*) [10], [34].

Fig. 1 shows a fragment of a goal model for the meeting scheduling system. The goals appearing there are behavioral goals.

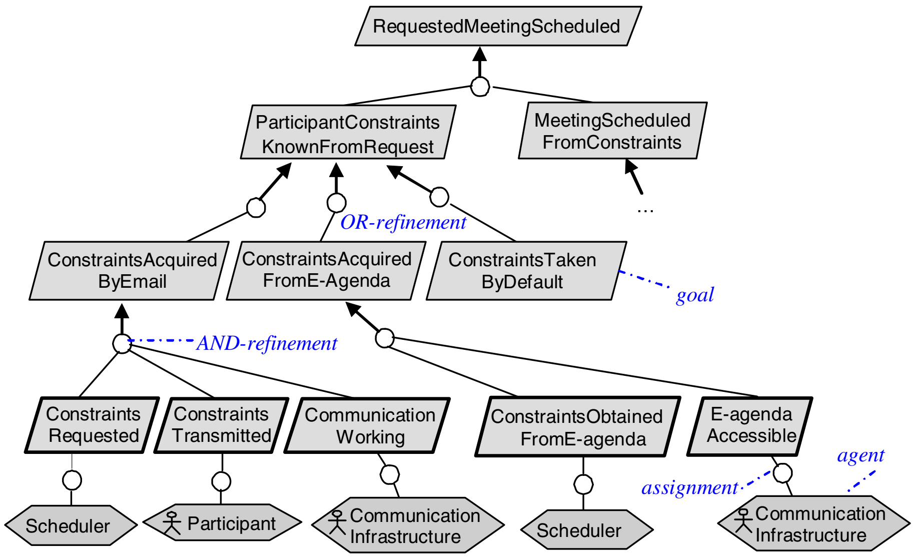

RequestedMeetingScheduled
ParticipantConstraints KnownFromRequest
MeetingScheduled FromConstraints
...
OR-refinement
ConstraintsAcquired ByEmail
ConstraintsAcquired FromE-Agenda
ConstraintsTaken ByDefault
goal
AND-refinement
Constraints Requested
Constraints Transmitted
Communication Working
ConstraintsObtained FromE-agenda
E-agenda Accessible
agent
assignment
Scheduler
Participant
Communication Infrastructure
Scheduler
Communication Infrastructure

Fig. 1. Goal model fragment for the meeting scheduling system

In a goal model, the top goals are the highest-level ones to be still in the system scope whereas the bottom goals are requirements or expectations assignable to single system agents (see the bottom of Fig. 1). In such graph, an AND-refinement link relates a goal to a set of subgoals called *refinement.* Domain properties may also be included in a refinement. The meaning is that the parent goal can be satisfied/satisficed by satisfying/satisficing all subgoals in the refinement, assuming the domain properties to hold. A goal node can be OR-refined into multiple AND-refinements; each of these is called *alternative* for satisfying/satisficing the parent goal. The meaning of multiple alternative refinements is that the parent goal can be satisfied/satisficed by satisfying/satisficing the conjoined subgoals in any of the alternative refinements.

For example, the goal ParticipantConstraintsKnownFromRequest in Fig. 1 is OR-refined into three alternatives: ConstraintsAcquiredByEmail, ConstraintsAcquiredFromE-Agenda, and ConstraintsTakenByDefault. The goal ConstraintsAcquiredByEmail is in turn AND-refined into three subgoals: ConstraintsRequested, ConstraintsTransmitted, and Communication-Working.

A goal model may also document conflicts that were detected among two or more goals (see Fig. 2 in the next section for an example). A *conflict* arises when behavioral goals cannot be satisfied together or when one of the soft goals contributes negatively to the other goals.

Goal nodes in a goal model are annotated with individual features such as their name and precise specification in natural language, their type, category, priority level, elicitation source, etc. Such annotations act as placeholders for dedicated techniques used in the RE process [25]. For example, priority levels are used for conflict management and requirements prioritization.

The systems *as-is* and *to-be* can both be captured within the same model. The two versions share high-level goals and differ along OR-refinement branches of common parent goals. We can thereby capture multiple variants in a system family.

Goal models can be used for a wide variety of purposes [24], [25], including: the evaluation of alternative options; the structuring and documentation of satisfaction arguments; the checking of the correctness of goal refinements and operationalizations; model animation; the analysis of risks, security threats, and conflicts; requirements prioritization; traceability management; the derivation of software architecture drafts; and the semi-automated generation of the requirements document. In this paper we focus on the use of goal models as a basis for evaluating alternative options.

4 Qualitative Reasoning about Alternative Options

Throughout the RE process we need to explore alternative options of different types, as introduced in Section 2, and select best ones based on specific evaluation criteria. In a GORE framework, options may refer to alternative goal refinements, responsibility assignments, operationalizations, conflict resolutions, risk resolutions, and threat resolutions.

To compare options and select best ones, we need evaluation criteria. The great idea set forth by the NFR framework was to use the soft goals identified in the goal model as evaluation criteria [34]. Different alternatives contribute to different degrees of satisficing these soft goals. Although originally described in the context of alternative goal refinements, the qualitative evaluation technique in [34] does in fact work for the other types of options as well.

The general idea is to use qualitative estimations for assessing the positive or negative contribution of alternative options to the soft goals [34]. The aim is to determine, in each alternative, a qualitative degree of satisfaction of the top-level soft goals in the goal refinement graph; the option with best degrees of satisfaction is then selected.

To achieve this we need, for each alternative option, to:

- assess its positive or negative influence on each leaf soft goal in the model,
- propagate such influence bottom-up in the goal graph until we reach the top-level soft goals.

Let us make these steps more precise while seeing them in action on our meeting scheduling system. Fig. 2 shows a portion of a goal model with soft goal refinements and conflicts – e.g., the faster the constraint acquisition process or the fewer the interactions with a participant, the less accurate the acquired constraints with respect to the participant's actual constraints.

4.1 Assessing the Qualitative Contribution of Alternative Options to Leaf Soft Goals

We first need to qualitatively assess the extent to which each alternative contributes to the various leaf goals in soft goal refinement trees – e.g., "++" (very positively), "+" (positively), "-" (negatively), "--" (very negatively), "*n*" (neutral). This amounts to putting some *qualitative weight* on refinement and conflict links in the goal model.

Condider our meeting scheduler case study, for example. Fig. 1 highlighted three alternative options for knowing the participant's date constraints:

- acquiring them by email requests,
- acquiring them by access to the participant's electronic agenda,
- taking default constraints (such as working days only).

Table 1 shows the qualitative contribution of these three options to the four leaf soft goals in Fig. 2. For example, the option ConstraintsAcquiredByEmail might

Table 1. Qualitative contributions of options to leaf soft goals

Leaf soft goals	**Alternative options**		
	Constraints Acquired By Email	Constraints Acquired FromE-Agenda	Constraints aken ByDefault
AccurateConstraints	+	-	-
FastConstraintAcquisition	-	+	+
Minimum Replanning	+	-	-
Minimum Interaction ForGettingConstraints	-	+	-

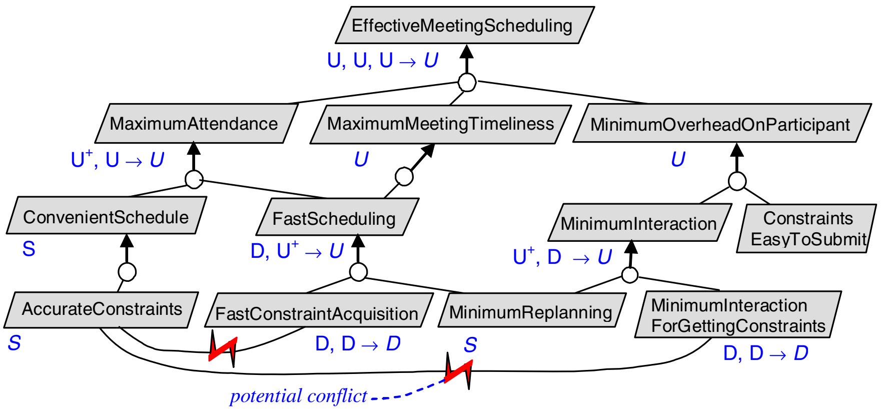


Fig. 2. Upward propagation of satisficing labels in a soft goal refinement graph [25]

contribute negatively to the soft goal FastConstraintAcquisition (as an invited participant might be non-responsive), positively to the soft goal AccurateConstraints (as the participant is likely to know her actual constraints), negatively to the soft goal MinimumInteractionForGettingConstraints (as the participant might get multiple email reminders), etc. On the other hand, the option ConstraintsAcquiredFromE-Agenda might contribute positively to FastConstraintAcquisition, negatively to AccurateConstraints (as the participant's e-agenda is likely to inaccurately reflect her actual availabilities), and positively to MinimumInteractionForGettingConstraints (as no participant interaction is required) .

4.2 Bottom-Up Propagation of Qualitative Contributions

The next step consists of propagating such contributions upwards in the sof goal graph through refinement and conflict links. For this we may use a procedure that assigns qualitative labels to each node in the graph [34]. A node is labelled:

S (satisficed):	if it is satisficeable and not deniable,
D (denied):	if it is deniable but not satisficeable,
C (conflicting):	if it is both satisficeable and deniable,
U (undetermined):	if it is neither satisficeable nor deniable.

The procedure propagates such labels bottom-up along refinement links, marked as "+" or "++" according to the strength of the positive contribution, and along conflict links, marked as "-" or "--" according to the severity of the negative contribution. Additional label values can be assigned at intermediate stages of the procedure, e.g.,

U^+: unconclusive positive support, U^-: unconclusive negative support,
?: user intervention required for getting an adequate label value.

An upward propagation step from offspring nodes to their parent node goes as follows.

1. The individual effect of a weighted link from an offspring to its parent is determined according to propagation rules such as the ones shown in Table 2 (we only consider a few weights to make things simpler). One additional rule states that *if* a node is a refinement node and all its offspring nodes have a *S* label *then* this node gets a *S* label. A node will thus in general have multiple labels –one per incoming link.
2. The various labels thereby collected at a parent node are merged into one single label. The user is first asked to combine the *U+* and *U-* labels into one or more *S*, *D*, *C* and *U* labels. The result is then combined into a single label by choosing the minimal label according to the following ordering:

$$S, D \geq U \geq C.$$

This upward propagation step is applied recursively until the top-level soft goals get a single label.

Let us see how this technique works for the options in Fig. 1 and the soft goal model portion in Fig. 2. We consider the first option ConstraintsAcquiredByEmail in Table 1; it therefore gets a "S" label at the left bottom in Fig. 2. According to Table 2, its "+" contribution links to AccurateConstraints and MinimumReplanning yield a "S" label for these two leaf soft goals (as no other subgoal is shown for the latter goals in the goal model; otherwise they would get a "U+" instead, unless all other subgoals have a "S"). On the other hand, the "-" links from this first option to FastConstraintAcquisition and MinimumInteractionForGettingConstraints yield a "D" label for these two leaf soft goals in Fig. 2.

At the next step, the "S" label of AccurateConstraints yields a "S" label for ConvenientSchedule, through a "+" link, and a "D" label for FastConstraintAcquisition, through the "-" conflict link shown in Fig. 2. Based on the above ordering, the labels "D,D" on the latter goal get merged into a single "D".

At the next step, this "D" label on FastConstraintAcquisition gets propagated through a "+" link to the parent goal FastScheduling as a "D" label again whereas the latter goal also gets a "U^+" label through a "+" link from its offspring MinimumReplanning (see Table 2). After user intervention this "U^+" might become "U" and the resulting label merge yields, according to the predefined label ordering, a single "U" label for FastScheduling. The process goes on upwards until we obtain "U, U, U" labels for the top soft goal EffectiveMeetingScheduling, which get merged into a single "U" label.

Table 2. Qualitative label propagation rules [34]

	Link weight		
	+	**-**	**n**
Offspring label	**Parent label**		
S	U+	D	U
D	D	U+	U
C	?	?	U
U	U	U	U

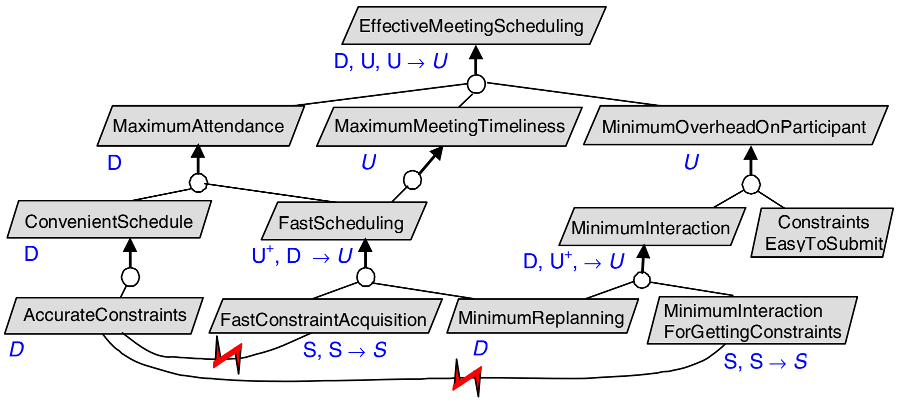


Fig. 3. Propagation of degrees of satisficing for the option Constraints Acquired From E-Agenda

A similar upward propagation for the second option in Table 1, namely, ConstraintsAcquiredFromE-Agenda, yields "D, U, U" labels for the top soft goal, as shown in Fig. 3.

4.3 Discussion

Overall the two options ConstraintsAcquiredByEmail and ConstraintsAcquiredFromE-Agenda seem comparable as they get the same top label after merge, namely, "U" (undetermined). However, the second option has a "D" label (denied) among the top labels which might make it less preferred. The third option of just taking default constraints gets one "D" more at the top which might be a good reason for discarding it.

This simple meeting scheduling example illustrates some limitations of qualitative approaches:

- The propagation rules make labels become rapidly unconclusive as we move up in the soft goal refinement tree. To overcome this, we might refine the qualitative labels, weights, and propagation rules in Table 2 to make them less rough.
- Still, the various types of labels and link weights have no clear meaning in terms of system-specific phenomena. Qualitative reasoning schemes provide some quick and cheap means for rough evaluation in the early stages of the RE process. Their applicability for effective decision support based on accurate arguments appears more questionable.
- All leaf soft goals used as evaluation criteria are considered to have the same importance. This is rarely the case in practice. For example, the soft goal AccurateConstraints is much more important than the soft goal MinimumInteractionForGettingConstraints in view of the key concern of maximum attendance to the meeting.

Regarding the second limitation above, the problem partly arises from the way soft goals are specified. Their specification often violates a basic RE principle stating that goals, requirements and assumptions should be *measurable*. The specification of a soft goal should therefore be complemented with a *fit criterion* that quantifies the extent to which this goal must be satisfied [39]. For example:

ConvenientSchedule: *The scheduled meeting dates shall meet the date constraints of invited participants as much as possible.*

Fit criterion: *Scheduled dates should fit all actual date constraints of at least 90% of invited participants.*

MinimumInteractionforGettingConstraints: *There should be as little interaction as possible with participants for getting their time constraints.*

Fit criterion: *There should be at most 4 interactions per participant to organize a meeting.*

Measurable soft goal specifications open the way to more accurate evaluation as we discuss now.

5 Lightweight Quantitative Evaluation of Alternative Options

To address the preceding limitations, we may use quantitative estimations for assessing the positive or negative contribution of alternative options to soft goals. The aim is to determine the overall score of each option with respect to all the leaf soft goals in the goal refinement graph, taking their respective importance into account. The option with highest score is then selected.

To achieve this,

- We assign different weights to the leaf soft goals in order to reflect their relative importance.
- We numerically score each option against the leaf soft goals. The scores should be grounded on measurable system phenomena related to the fit criteria of these soft goals.
- We collect the weights and scores in a weighted matrix for overall comparison.

5.1 Quantifying Option Contributions to Leaf Soft Goals through Score Matrices

Weighted matrices are a standard system engineering technique for quantitative decision support. Such matrices bear similarities with those used for risk management within the RE process [14]. They capture estimated scores of each option with respect to the evaluation criteria used.

- As our evaluation criteria are the soft goals from the goal model, we first assign a weight to each leaf soft goal to reflect its importance relatively to others – typically, a numerical proportion. Such weight can be derived from the goal's priority level specified in the goal model [25].
- A matrix cell associated with an option *opt* and a leaf soft goal *lsg* captures the estimated score of the option with respect to this soft goal. A score *X* means that the option is estimated to satisfy the soft goal in *X%* of the cases.

- The last row of the matrix gives the overall score of each option as a weighted summation of its scores with respect to each leaf soft goal:

$$totalScore\,(opt) = \sum_{lsg} (Score\,(opt, lsg) \times Weight\,(lsg))$$

Table 3. Weighted matrix for evaluating alternative options against all leaf soft goals [25]

Leaf soft goals	*Importance weighting*	**Option scores**		
		Constraints Acquired By Email	*Constraints Acquired FromE-Agenda*	*Constraints Taken ByDefault*
AccurateConstraints	*0.50*	*0.90*	*0.30*	*0.10*
Fast ConstraintAcquisition	*0.30*	*0.50*	*0.90*	*1.00*
MinimumReplanning	*0.10*	*0.80*	*0.30*	*0.10*
MinimumInteraction ForGettingConstraints	*0.10*	*0.50*	*1.00*	*1.00*
TOTAL	**1.00**	**0.73**	**0.55**	**0.46**

Table 3 shows such quantitative evaluation for the options and soft goals evaluated qualitatively in Table 1. In this evaluation, the soft goal AccurateConstraints is considered relatively much more important than the soft goals MinimumInteractionForGettingConstraints and MinimumReplanning (the latter having the same level of limited importance). The first option, ConstraintsAcquiredByEmail, is estimated to satisfy the soft goal AccurateConstraints in *90 %* of the cases –as invited participants are expected to directly express their own constraints. The 10% remaining stand for participants confusing dates or having taken other commitments in the meantime. Overall, this first option outperforms the others in view of the relative weights assigned to each soft goal.

For this approach to work effectively, we need to be able to determine *adequate* option scores against each leaf soft goal in the goal model. Note that the accuracy of an individual score taken in isolation is not what matters the most. We can draw meaningful comparative conclusions as long as *all scores are set consistently, from one alternative to the other, as a common basis for making comparisons*. Nevertheless, to avoid subjective estimations resulting in questionable decisions, the scores should be grounded on domain expertise, experience with similar systems and interpretations in terms of measurable phenomena in the system.

5.2 Deriving Option Scores from Measures of Soft Goal Satisficing

The goal model may be used to estimate option scores that are grounded on measurable system phenomena. The general idea is to: identify gauge variables referred to in the specification of the soft goals and their fit criterion; evaluate such variables along the refinement tree of the various options; and derive options scores from the values obtained.

Identifying and evaluating gauge variables. A *gauge variable* is a variable associated with a specific leaf soft goal in the goal model. It may capture:

- a quantity the soft goal prescribes to *Improve*, *Increase*, *Reduce*, *Maximize*, or *Minimize*;
- the estimated cost of satisficing this soft goal;
- the estimated time taken for satisficing it.

Consider the leaf soft goal *MinimumInteractionForGettingConstraints* in the meeting scheduling system. Its specification was seen to be:

There should be as little interaction as possible with participants for getting their time constraints.

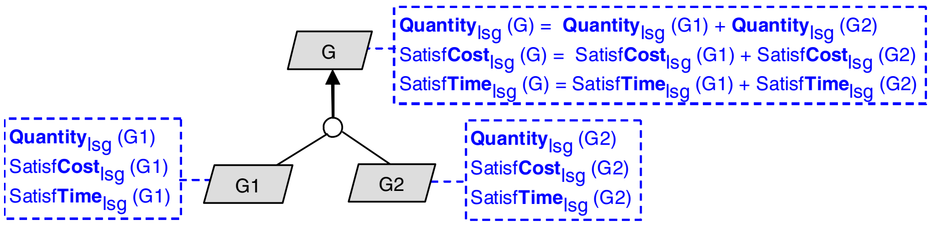
Fit criterion*: There should be at most 4 interactions per participant to organize a meeting.*

The variable ExpectedNumberOfInteractions may be derived from this specification as a gauge variable for this soft goal. It estimates the average number of interactions between a participant and the scheduler to get the participant's constraints. Note that the full specification implicitly specifies an *ideal target value* (*0*) and an *acceptability threshold* (*4*).

To support the evaluation of options based on soft goals grounded on measurable phenomena, a gauge variable should meet the following requirements.

- *Soft goal measure:* To enable comparisons of options with respect to soft goals, the variable should provide some measure of the degree of satisficing of its associated leaf soft goal.
- *Cumulative quantity:* To enable accurate estimations of its values for the different options, the variable should propagate additively along the AND-trees refining these options in the goal model.

Fig. 4 explains what is meant by additive propagation. Let *lsg* denote a leaf soft goal and *gv* a cumulative gauge variable associated with it. In view of the semantics of goal AND-refinement, the value of *gv* at a parent goal *G* in the refinement tree of a specific option is obtained by summing up the values of *gv* at the subgoals *G1* and *G2*. When a value for the variable at some subgoal makes no sense, we just ignore it in the summation.



G
G1
G2
$\textbf{Quantity}_{lsg}$ (G) = $\textbf{Quantity}_{lsg}$ (G1) + $\textbf{Quantity}_{lsg}$ (G2)
Satisf$\textbf{Cost}_{lsg}$ (G) = Satisf$\textbf{Cost}_{lsg}$ (G1) + Satisf$\textbf{Cost}_{lsg}$ (G2)
Satisf$\textbf{Time}_{lsg}$ (G) = Satisf$\textbf{Time}_{lsg}$ (G1) + Satisf$\textbf{Time}_{lsg}$ (G2)
$\textbf{Quantity}_{lsg}$ (G1)
Satisf$\textbf{Cost}_{lsg}$ (G1)
Satisf$\textbf{Time}_{lsg}$ (G1)
$\textbf{Quantity}_{lsg}$ (G2)
Satisf$\textbf{Cost}_{lsg}$ (G2)
Satisf$\textbf{Time}_{lsg}$ (G2)

Fig. 4. Cumulative propagation of gauge variables [25]

The merits of an option are obtained by upward propagation, along the option's refinement tree, of the cumulative values of the gauge variables measuring the satisficing of the leaf soft goals in the goal model, starting from the option's own leaf subgoals. The reason for performing such up-propagation is that the values of gauge variables will generally be more easily and accurately estimated for the finer-grained leaf subgoals of the option.

Table 4 illustrates the evaluation of gauge variables by up-propagation from the leaf subgoals in the various options. Each gauge variable there corresponds to a quantity that the associated leaf soft goal in Table 3 prescribes to *Maximize* or *Minimize.*

Table 4. Values of gauge variables for soft goal satisficing by alternative options [25]

Soft-goal gauge variable	**Option values**		
	Constraints Acquired By Email	*Constraints Acquired FromE-Agenda*	*Constraints Taken ByDefault*
Expected Number of Correct Free Half-Days per Week	*9*	*3*	*1*
Expected Constraint Acquisition Time (in days)	*3*	*1*	*0*
Expected Number of Replannings	*0.5*	*2*	*4*
Expected Number of Interactions	*2*	*0*	*0*

For example, consider the gauge variable *ExpectedNumberOfInteractions* in Table 4, associated with the soft goal MinimumInteractionForGettingConstraints in Table 3. The estimated value "2" for this variable in the option *ConstraintsAcquiredByEmail* is obtained by summing the value "1" for the subgoal ConstraintsRequested in Fig. 1 (one interaction per requested constraints) and the value "1" for the subgoal ConstraintsTransmitted (another interaction per returned constraints). The estimated value "3" in the same column, for the same option and the gauge variable *ExpectedConstraintAcquisitionTime* associated with the soft goal FastConstraintAcquisition, results from "0" (time taken for the scheduler to request constraints) + "3" (estimated average time, in days, for a participant to return her constraints). Each gauge variable in Table 4 is derived from the specification and fit criterion of the corresponding leaf soft goal in Table 3; its values are obtained by up-propagation along the refinement tree of the corresponding option from the estimated values at leaf nodes (see Fig. 4).

Deriving option scores from values of gauge variables. Once the overall gauge values are obtained at the root of the refinement tree of each option by such up-propagation, we can derive the scores of each option as follows.

Let *Score (opt, lsg)* denote the score of option *opt* with respect to the leaf soft goal *lsg*. Let g_V denote the value of the gauge variable associated with *lsg*, obtained at the root of *opt*'s refinement tree by up-propagation from its leaf subgoals. Let g_T denote the ideal target value we would expect for this variable and g_{max} its maximum acceptable value; g_T and g_{max} are derived from *lsg*'s specification and fit criterion, respectively. We then have:

$$Score\,(opt,\,lsg) = 1 - (\,|\,g_T - g_V\,|\,)\,/\,g_{max}$$

As we can see from this formula, the closer the gauge value to its ideal target relatively to its maximum acceptable value, the closer the corresponding score to 1. The more distant the gauge value from its ideal target relatively to its maximum acceptable value, the closer the corresponding score to 0.

Table 5. Evaluation of alternative options against leaf soft goals based on gauge variables

Leaf soft goals	*Importance weighting*	**Option scores**				
		g_T	g_{max}	*Constraints Acquired ByEmail*	*Constraints Acquired FromE-Agenda*	*Constraints Taken ByDefault*
AccurateConstraints	*0.50*	10	10	*0.90*	*0.30*	*0.10*
Fast ConstraintAcquisition	*0.30*	0	6	*0.50*	*0.84* (0.90)	*1.00*
MinimumReplanning	*0.10*	0	4	*0.87* (0.90)	*0.50* (0.30)	*0* (0.10)
Minimum InteractionFor GettingConstraints	*0.10*	0	4	*0.50*	*1.00*	*1.00*
TOTAL	**1.00**			**0.74** (0.73)	**0.55**	**0.45** (0.46)

Table 5 shows another weighted matrix for evaluating alternative options in our meeting scheduling system. Compared to the matrix in Table 3, this one is based on such score derivation from the values of the gauge variables in Table 4. (The numbers in parentheses refer to the corresponding rough estimations in Table 3.)

For example, Table 4 gave a value of "2" for the gauge variable ExpectedNumberOfInteractions in the option ConstraintsAcquiredByEmail. This gauge measures the degree of satisficing of the *Minimize* soft goal MinimumInteractionForGettingConstraints. From the specification and fit criterion of this soft goal, we get the values "0" for g_T and "4" for g_{max}. As the value g_V for this gauge, obtained by up-propagation from the leaf nodes in the refinement tree of the option ConstraintsAcquiredByEmail, is "2", the score obtained according to the preceding formula is "0.50". Similarly, Table 4 gave a value of "9" for the gauge variable ExpectedNumberOfCorrectFreeHalfDaysPerWeek in the option ConstraintsAcquiredByEmail. This gauge measures the degree of satisficing of the *Maximize* soft goal AccurateConstraints; a value *X* for this variable means that, among the 10 working half-days in a week where a participant is *stated* to be free, *X* of these are half-days where she is *actually* free. From the specification and fit criterion of this soft goal, we get the values "10" for g_T and "10" for g_{max}. As the value g_V for this gauge, obtained by up-propagation from the leaf nodes in the refinement tree of the option ConstraintsAcquiredByEmail, is "9", the score obtained according to the preceding formula is "0.90".

In comparison with Table 3, the numbers in Table 5 are not significantly different. They result in the same comparative evaluation yielding the selection of the first option ConstraintsAcquiredByEmail as best one. There are significant differences, however, in the arguability of conclusions and the way we get to them through gauge variables:

- these conclusions rely on measures of degree of soft goal satisficing that are based on system phenomena;
- they are derived systematically from the specifications of the leaf soft goals in the goal model.

We therefore gain increased confidence in the adequacy of our conclusions.

A more sophisticated approach to quantitative reasoning about alternative options can be found in [30]. Quality variables are used there instead of gauge variables. They are random variables with probability distribution functions. The analysis then is more accurate but more heavyweight as a price to pay.

6 Conclusion

The evaluation of alternative system options is at the heart of the RE process. John Mylopoulos has significantly contributed to the development of concepts, models and techniques for this critical task. The important role played by goal models, soft goals as evaluation criteria, and propagation of positive/negative goal contributions are now much better understood. Others have built upon his results and will continue to explore the directions he has opened.

Beyond the work outlined in this paper, the RE community owes much to John for his contribution to raising the technical standards in the field, his open-minded and interdisciplinary attitude in research, his humility and friendliness in research interactions, and the network of colleagues he has created worldwide.

Acknowledgement. The principle of using measurable fit criteria for quantitative reasoning about soft goals results from joint work with Emmanuel Letier. Warm thanks are due to the reviewer of this paper whose requests for clarification resulted in the introduction of quantified links between option scores and soft-goal gauge variables.

References

1. Alford, M.: A Requirements Engineering Methodology for Real-Time Processing Requirements. IEEE Transactions on Software Engineering 3(1), 60–69 (1977)
2. Bell, T.E., Thayer, T.A.: Software Requirements: Are They Really a Problem? In: Proc. ICSE 1976: 2nd International Conference on Software Enginering, San Francisco, pp. 61–68 (1976)
3. Boehm, B.W.: Software Engineering Economics. Prentice-Hall, Englewood Cliffs (1981)
4. Borgida, A., Mylopoulos, J., Reiter, R.: And Nothing Else Changes: The Frame Problem in Procedure Specifications. In: Proc. ICSE 1993 - 15th International Conference on Software Engineering, Baltimore (May 1993)

5. Brodie, M., Mylopoulos, J., Schmidt, J. (eds.): On Conceptual Modeling: Perspectives from Artificial Intelligence, Databases, and Programming Languages. Springer, Heidelberg (1984)
6. Brooks, F.P.: No Silver Bullet: Essence and Accidents of Software Engineering. IEEE Computer 20(4), 10–19 (1987)
7. Castro, J., Kolp, M., Mylopoulos, J.: Towards Requirements-Driven Information Systems Engineeering: the TROPOS Project. Information Systems 27, 365–389 (2002)
8. Chung, L., Nixon, B., Yu, E., Mylopoulos, J.: Non-functional requirements in software engineering. Kluwer Academic, Boston (2000)
9. Damas, C., Lambeau, B., van Lamsweerde, A.: Scenarios, Goals, and State Machines: A Win-Win Partnership for Model Synthesis. In: Proceedings of FSE 2006, 14th ACM International Symp. on the Foundations of Software Engineering, November 2006, Portland, OR (2006)
10. Dardenne, A., Fickas, S.S., van Lamsweerde, A.: Goal-directed Concept Acquisition in Requirements Elicitation. In: Proc. 6th International Workshop on Software Specification and Design, Como, Italy, pp. 14–21 (1991)
11. Dardenne, A., van Lamsweerde, A., Fickas, S.: Goal-Directed Requirements Acquisition. Science of Computer Programming 20, 3–50 (1993)
12. Darimont, R., van Lamsweerde, A.: Formal Refinement Patterns for Goal-Driven Requirements Elaboration. In: FSE'4 - Fourth ACM SIGSOFT Symp. on the Foundations of Software Engineering, October 1996, pp. 179–190 (1996)
13. DeMarco, T.: Structured Analysis and System Specification. Yourdon Press (1978)
14. Feather, M.S., Cornford, S.L.: Quantitative Risk-Based Requirements Reasoning. Requirements Engineering Journal 8(4), 248–265 (2003)
15. Fuxman, A., Pistore, M., Mylopoulos, J., Traverso, P.: Model Checking Early Requirements Specifications in Tropos. In: Proc. RE 2001 - 5th Intl. Symp. Requirements Engineering, Toronto (August 2001)
16. Greenspan, S.J., Mylopoulos, J., Borgida, A.: Capturing More World Knowledge in the Requirements Specification. In: Proc. ICSE-1982: 6th Intl. Conf. on Software Enginering, Tokyo (1982)
17. Greenspan, S.J., Borgida, A., Mylopoulos, J.: A Requirements Modeling Language and its Logic. Information Systems 11(1), 9–23 (1986)
18. Heninger, K.L.: Specifying Software Requirements for Complex Systems: New Techniques and their Application. IEEE Transactions on Software Engineering 6(1), 2–13 (1980)
19. Hui, B., Laiskos, S., Mylopoulos, J.: Requirements Analysis for Customizable Software: A Goals Skills Preferences Framework. In: Proc. RE 2003: 11th IEEE Joint Intl. Requirements Engineering Conference, Monterey, CA, September 2003, pp. 117–126 (2003)
20. Jackson, M.: Software Requirements & Specifications - A Lexicon of Practice, Principles and Prejudices. ACM Press, Addison-Wesley (1995)
21. van Lamsweerde, A., Dardenne, A., Delcourt, B., Dubisy, F.: The KAOS Project: Knowledge Acquisition in Automated Specification of Software. In: Proc. AAAI Spring Symposium Series, Track: Design of Composite Systems, Stanford University, American Association for Artificial Intelligence, March 1991, pp. 59–62 (1991)
22. van Lamsweerde, A.: Requirements Engineering in the Year 00: A Research Perspective. In: Proc. ICSE 2000: 22nd International Conference on Software Engineering, pp. 5–19. ACM Press, New York (2000) (invited keynote paper)
23. van Lamsweerde, A.: Elaborating Security Requirements by Construction of Intentional Anti-Models. In: Proc. ICSE 2004, 26th Intl. Conference on Software Engineering, Edinburgh, May 2004, pp. 148–157. ACM-IEEE (2004)

24. van Lamsweerde, A.: Requirements Engineering: From Craft to Discipline. Invited Paper for the ACM Sigsoft Outstanding Research Award, Proc FSE-16: 16th ACM Conference on the Foundations of Software Engineering, Atlanta (November 2008)
25. van Lamsweerde, A.: Requirements Engineering: From System Goals to UML Models to Software Specifications, January 2009. Wiley, Chichester (2009)
26. van Lamsweerde, A., Darimont, R., Letier, E.: Managing Conflicts in Goal-Driven Requirements Engineering. IEEE Transactions on Sofware Engineering 24(11), 908–926 (1998)
27. van Lamsweerde, A., Letier, E.: Handling Obstacles in Goal-Oriented Requirements Engineering. IEEE Transactions on Software Engineering 26(10), 978–1005 (2000)
28. Letier, E., van Lamsweerde, A.: Agent-Based Tactics for Goal-Oriented Requirements Elaboration. In: Proc. ICSE 2002: 24th Intl. Conf. on Software Engineering, May 2002. ACM-IEEE (2002)
29. Letier, E., van Lamsweerde, A.: Deriving Operational Software Specifications from System Goals. In: Proc. FSE'10: 10 th ACM Symp. Foundations of Software Engineering, Charleston (November 2002)
30. Letier, E., van Lamsweerde, A.: Reasoning about Partial Goal Satisfaction for Requirements and Design Engineering. In: Proc. FSE 2004, 12th ACM International Symp. on the Foundations of Software Engineering, Newport Beach, CA, November 2004, pp. 53–62 (2004)
31. Liu, L., Yu, E., Mylopoulos, J.: Security and Privacy Requirements Analysis within a Social Setting. In: Proc. RE 2003: Intl. Requirements Engineering Conf. (September 2003)
32. Mylopoulos, J.: Information Modeling in the Time of the Revolution, Invited Review. Information Systems 23(3-4), 127–155 (1998)
33. Mylopoulos, J., Bernstein, P., Wong, H.: A Language Facility for Designing Interactive Database-Intensive Applications. ACM Transactions on Database Systems 5(2) (June 1980)
34. Mylopoulos, J., Chung, L., Nixon, B.: Representing and Using Nonfunctional Requirements: A Process-Oriented Approach. IEEE Transactions on Sofware Engineering 18(6), 483–497 (1992)
35. Mylopoulos, J., Chung, L., Liao, S., Wong, H., Yu, E.: Exploring Alternatives During Requirements Analysis. IEEE Software 18(1), 92–96 (2001)
36. Mylopoulos, J., Chung, L.L., Yu, E.E.: From Object-Oriented to Goal-Oriented Requirements Analysis. Communications of the ACM 42(1), 31–37 (1999)
37. Parnas, D.L., Madey, J.: Functional Documents for Computer Systems. Science of Computer Programming 25, 41–61 (1995)
38. Ponsard, C., Massonet, P., Molderez, J.F., Rifaut, A., van Lamsweerde, A.: Early Verification and Validation of Mission-Critical Systems. Formal Methods in System Design 30(3), 233–247 (2007)
39. Robertson, S., Robertson, J.: Mastering the Requirements Process. ACM Press, Addison-Wesley (1999)
40. Ross, D.T., Schoman, K.E.: Structured Analysis (SA): A Language for Communicating Ideas. IEEE Transactions on Software Engineering 3(1), 16–34 (1977)
41. Roussopoulos, N., Mylopoulos, J.: Using Semantic Networks for Database Management. In: Proc. 1st Conf. on Very Large Databases (VLDB), September 1975, pp. 144–172 (1975)
42. Tran Van, H., van Lamsweerde, A., Massonet, P., Ponsard, C.: Goal-Oriented Requirements Animation. In: Proc. RE 2004, 12th IEEE Joint International Requirements Engineering Conference, Kyoto, September 2004, pp. 218–228 (2004)
43. Yu, E.S.K.: Modelling Organizations for Information Systems Requirements Engineering. In: Proc. RE 1993 - 1st Intl Symp. on Requirements Engineering, pp. 34–41 (1993)

Supporting Requirements Elicitation through Goal/Scenario Coupling

Colette Rolland and Camille Salinesi

Centre de Recherche en Informatique, 90,
rue Tolbiac 75013 Paris, France
{colette.rolland,camille.salinesi}@univ-paris1.fr

Abstract. Goals have long been recognized to be an essential component involved in the Requirements Engineering (RE) process. They have proved to be an effective way to support a number of requirements engineering activities such as requirements elicitation, systematic exploration of design choices, checking requirements completeness, ensuring requirements pre-traceability and helping in the detection of threats, conflicts, obstacles and their resolution. The leading role played by goals in the RE process led to a whole stream of research on goal modeling, goal specification/formulation and goal-based reasoning for the multiple aforementioned purposes. On the other hand, there is evidence that dealing with goal is not an easy task and presents a number of difficulties in practice. To overcome these difficulties, many authors suggest combining goals and scenarios. The reason is that they complement each other: there is a mutual mitigation of difficulties encountered with one by using the other. The paper reviews various research efforts undertaken in this line of research. It uses L'Ecritoire, an approach that guides the requirements elicitation and specification process through interleaved goal modelling and scenario authoring to illustrate the combined use of goals and scenarios to reason about requirements for the system To-Be.

1 Introduction

Goals have long been recognized to be an essential component involved in the Requirements Engineering (RE) process. In their seminal paper, Ross and Schoman stated "requirements definition must say why a system is needed, based on current and foreseen conditions, which may be internal operations or external market. It must say what a system features will serve and satisfy this context. And it must say how the system is to be constructed" [1]. Typically, the current system is analyzed; problems are pointed out and opportunities are identified; high level strategic goals are elicited and refined to address such problems and meet such opportunities; requirements are then elaborated to meet these goals. Goals are thus, the driving force of the requirements engineering process.

Goals have proved to be an effective way to elicit requirements [2] [3] but also to support a systematic exploration of design choices [4] [5] [6] to check requirements completeness [7], to ensure requirements pre-traceability [8] [9] to help in the detection of threats [10] conflicts [11] and obstacles [12] and their resolution, and to analyse systems evolution [18] [27]. The leading role played by goals in the RE process led to a whole stream of research on goal modeling, goal specification/formulation and goal-based reasoning for the multiple aforementioned purposes. John Mylopoulos and his team played a leading role in this stream and produced a vast set of papers that are highly referenced in the RE literature [43], [44], [45], [46], [47], [48], [49], [50], [51], [52].

However, several authors [13], [14], [3], [15] also acknowledge the fact that dealing with goal is not an easy task. Experience shows that it is difficult for domain experts to deal with the fuzzy concept of a goal [16] [17]. Though it is assumed that systems are constructed with some goals in mind [7], experience [18] shows that goals are not given and therefore the question of where they originate from [1] acquires importance. In addition, the goals often given by organizations are not real ones but reflect an idealized view of the enterprise. Therefore, proceeding from these may lead to ineffective requirements [19]. Thus, goal discovery is rarely an easy task.

Scenario modelling is another way to facilitate requirements elicitation from an analysis of the wider context in which the system will operate. By capturing examples and illustrations, scenarios help people in reasoning about complex systems [27]. A scenario is ‘a possible behavior limited to a set of purposeful interactions taking place among several agents’ [20] [21]. It describes a desirable functionality of a system under design, and thus, helps in identifying requirements [22]. The problem with scenarios is that they are inherently partial and therefore they raise a coverage problem making it impossible to verify the completeness of the requirements [23]. Besides, because they deal with examples and illustrations, scenarios only provide restricted requirements descriptions which need to be generalized to obtain complete requirements [24].

Some proposals have been made to couple goals and scenarios together. Potts [19] [25] claims that it is «unwise to apply goal based requirements methods in isolation» and suggests that they should be complemented with scenarios. Yet other proposals exist which interpret scenarios as containing information on how goals can be achieved [26] [23]. Thus, the goal-scenario combination has been used to operationalize goals. Yet others look upon goals as playing a documentation role only. This view is taken in [28] [29] [30] [31] [53] where a goal is considered as a contextual property of a use case (integrated set of scenarios) i.e. it is a property that relates the scenario to its organizational context. Cockburn [23] goes beyond this view and suggests the use of goals to structure use cases by connecting every action in a scenario to a goal of another use case at a lower level of abstraction. In this sense a scenario is discovered each time a goal is.

All these views suggest a unidirectional relationship between goals and scenarios: the goal-scenario combination has been used to operationalize goals. This is because scenarios can be interpreted as containing information on how goals can be achieved. In the CREWS project [3], we developed the L’Ecritoire approach in which we view this relationship as bi-directional: just as goals can help in scenario discovery, so also scenarios can help in goal discovery. The total solution is in two parts. First, for a goal, textual scenarios are authored. Thereafter, the authored scenario is explored to yield goals which, in turn, cause new scenarios to be authored and so on. The rest of this paper illustrates how this bidirectional coupling can help eliciting requirements.

The paper is organized in two main sections. The first presents an overview of L’Ecritoire approach: the goal-scenario coupling, the notion of a requirement chunk, scenario authoring, goal discovery and associated enactable rules. In the third section, the functionality of the tool that supports the approach is illustrated through a usage scenario. The concluding section highlights the benefits of goal-scenario coupling.

2 Overview of the L'Ecritoire Approach

L'Ecritoire is an approach supported by a software tool for requirements elicitation, structuration, and documentation. It draws heavily from the work on goal, scenario, and goal-scenario coupling found in the literature.

2.1 The Notion of a Requirement Chunk

At the core of the L'Ecritoire approach is the notion of a Requirement Chunk. We define a Requirement Chunk (RC) as a pair <G, Sc> where G is a goal and Sc is a scenario. Since a goal is intentional and a scenario is operational in nature, a requirement chunk is a possible way in which the goal can be achieved.

- A goal is defined [32] as 'something that some stakeholder hopes to achieve in the future'. In L'Ecritoire, it is expressed as a clause with a main verb and several parameters, where each parameter plays a different role with respect to the verb. For example in the goal statement :

  'Withdraw verb (cash)target (from ATM)means'

  'Withdraw' is the main verb, 'cash' is the parameter target of the goal, and 'from ATM' is a parameter describing the means by which the goal is achieved. We adopt the linguistic approach of Fillmore's Case grammar [33], and its extensions [34] [35], to define goal parameters [36]. Each type of parameter corresponds to a case and plays a different role within a goal statement. One of these parameters is the *manner* which refers to an approach, a way by which the goal can be achieved. We will see later that L'Ecritoire uses this parameter to distinguish two RCs corresponding to two different ways to achieving the same goal. For example: *Withdraw cash from the ATM normally* and *Withdraw cash from the ATM with insufficient account balance .Normally* and *with insufficient account balance* are two different manners for Withdrawing cash from an ATM.
- A scenario is 'a possible behavior limited to a set of purposeful interactions taking place among several agents' [32]. In L'Ecritoire, a scenario is defined as composed of one or more actions which describe a unique path leading from an initial to a final state.
- The initial state defines the preconditions for the scenario to be triggered. For example, the scenario 'Withdraw cash from the ATM' cannot be performed if the initial state 'The bank customer has a card' and 'The ATM is ready' is not true. The final state is the state reached at the end of the scenario. The scenario 'Withdraw cash from the ATM' leads to the compound state 'The user has cash', and 'The ATM is ready'.
- Actions in a scenario are of two types, atomic actions and flows of actions. Atomic actions are interactions 'from' an agent 'to' another which affect some 'parameter objects'. The clause 'The bank customer inserts a card in the ATM' is

an example of an atomic action involving two different agents 'The bank customer' and 'the ATM' and having the ‘card’ as parameter.

Flows of actions are composed of several actions and can be of different types: sequence, concurrent, iterative and conditional. The sentence 'The bank customer gets a card from the bank, then the bank customer withdraws cash from the ATM' is an example of a sequence comprising two atomic actions. The flow of actions 'While the ATM keeps the card, the ATM displays an "invalid card" message to the bank customer' is concurrent; there is no predefined order between the two concurrent actions.

- Requirement chunks can be assembled together through *AND*, *OR* and *Refined-by* relationships. *ORed* RCs are variants one of the others; they represent alternative ways of fulfilling the same goal. RCs related by a *Refined-by* relationship are at different level of abstraction. *ANDed* RCs are complementary chunks in the sense that every one requires the others to define a completely functioning system.
- The L’Ecritoire approach identifies three levels of requirements abstraction, namely the *contextual*, *system interaction* and *system internal* levels.

  The aim of the contextual level is to identify the services that a system should provide to fulfill a high level, strategic goal. At the system interaction level the focus is on the interactions between the system and its users to achieve the services assigned to the system at the contextual level. Thus, the contextual level is the bridge between strategic goals and system functional requirements. The system internal level focuses on what the system needs to perform the interactions selected at the system interaction level. The ‘what’ is expressed in terms of system internal actions that involve system objects but may require external objects such as other systems. This level defines the software requirements to meet the system functional requirements.

2.2 The Scenario-Authoring, Goal-Discovery Process

The L’Ecritoire requirements elicitation process is organized around two main activities:

- goal discovery and,
- scenario authoring

In this process, goal discovery and scenario authoring are complementary activities, the former following the latter. As shown in Figure 1, these activities are repeated to incrementally populate the requirement chunk hierarchy.

As shown in Fig.1, the requirements elicitation process is an iterative process comprising a number of steps by which the RCs at the three levels of abstraction (contextual, interaction and internal) are progressively discovered and specified. Each step exploits the goal-scenario relationship in both, the forward and backward directions. A step starts with a goal and the goal-scenario relationship is then exploited in the forward direction to author a scenario which is a possible concretization of this goal. Then the goal-scenario relationship is exploited in the reverse direction to discover new goals based on an analysis of the scenario. In subsequent steps, starting from the goals of these new RCs, scenarios are authored and the requirements elicitation cycle thus continues.

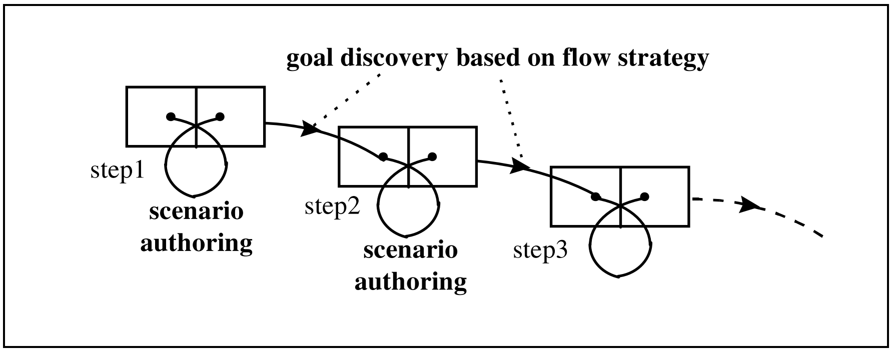


Fig. 1. The requirements elicitation process

Thus, the starting point can be a single goal at the contextual level for which a first scenario will be authored and goals discovered for which new scenarios will be authored etc.

Each of the two main activities, goal discovery and scenario authoring, is supported by systematic rules, (1) authoring rules and (2) discovery rules. Authoring rules allow L'Ecritoire scenarios which are textual to be authored. Discovery rules are for discovering goals through the analysis of authored scenarios.

Authoring rules and discovery rules that guide the iterative process are explained in detail in the following.

2.3 Authoring Rules

The role of authoring rules is to ensure the authoring of quality scenarios that can support the automatic goal discovery process. These rules combine style and contents guidelines with an advanced linguistic support embedded in a software tool that we refer to as linguistic device. The former help L'Ecritoire user to write scenarios whereas the latter help in scenario analysis, disambiguation and completion. The linguistic device is based on a case grammar and case patterns. A detailed description can be found in [37] [38] [39].

2.3.1 Style and Contents Guidelines

A L'Ecritoire scenario is a textual one. It is a full prose narrative describing a possible behavior to fulfill the goal of the associated RC. Style guidelines help the L'Ecritoire user in the wording of the text. "Make use of the active voice" and "Avoid the use of pronouns and articles" are examples of style guidelines. Contents guidelines advise on what is a correct text content. "Any communication action should be directed from an agent to another agent and apply on a parameter" is an example of contents guideline.

Style guidelines are applicable to any type of RCs, contextual, system interaction and system internal whereas there are specific contents guidelines for each level of abstraction. The violation of a style guideline is the sign of an incorrect scenario and can lead to erroneous results when applying the enactable rules. The second style guideline above helps for example, in removing ambiguity. Contents guidelines help in writing complete communication statements [41] [39] [37].

We refer to the scenario written by l'Ecritoire user in narrative prose with the help of style and contents guidelines, as the *initial narrative scenario.*

2.3.2 Linguistic Device

The linguistic device enables the transformation of the initial narrative scenario into a complete, non ambiguous text matching the L'Ecritoire scenario model. This transformation is required to produce a quality requirement document and, also to allow the automatic discovery of goals from scenario analysis using the discovery rules.

Linguistic approach. Scenario transformation is supported by a linguistic approach based on a Case Grammar inspired by Fillmore's Case Theory [33]. As shown in Figure 2, the approach is grounded in three elements, semantic structures, semantic patterns and scenario model. Semantic structures correspond to the linguistic structures of statements in the scenario text whereas semantic patterns provide the semantic meaning of these statements. According to Chomsky, linguistic structures are the surface structures of statements whereas semantic patterns correspond to their deep structures [40]. The scenario model provides the structure of concepts of any scenario as described in section 2. The correspondence between linguistic structures and semantic patterns helps in associating a meaning to a scenario statement; the correspondence between a semantic pattern and the scenario model defines the relationship between the textual form of a scenario and its conceptual form.

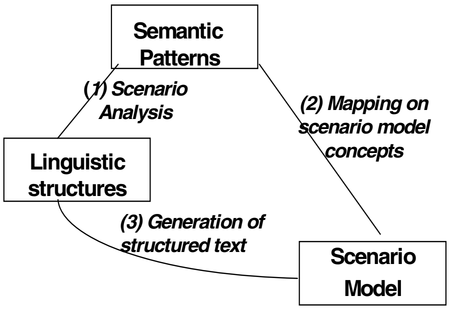


Fig. 2. The linguistic approach for scenario semantic analysis

The semantic analysis of the initial narrative scenario is performed in three steps. The scenario text is parsed and every clause and sentence is matched first onto linguistic structures and then onto semantic patterns (1); thus every clause and sentence of the scenario text is represented as semantic pattern instances. Elements of the pattern instances are mapped onto concepts of the scenario model (2) and finally, the scenario structured text is generated (3).

The example below illustrates the three steps:

Initial clause: ''A card is inserted by a user into the ATM''

Linguistic structure instance: ['A card'] (Subject)$_{\text{Object}}$ ['is inserted'](Main Verb)$_{\text{Communication}}$ ['by a user'](Complement)$_{\text{Agent+Source}}$

['into the ATM'](Complement)$_{\text{Destination}}$](VG passive)*Communication*

Semantic pattern (communication) instance: Communication (*'insert'*) [Agent: *'a user'*; Object: *'a card'*; Source: *'a user'*; Destination: *'the ATM'*]

Model concept (Atomic action)

Name: *'insert'*;
To Agent: *'the ATM'*;
From Agent: *'a user'*;
Parameter: *'a card'*;

Scenario linguistic completion. Linguistic incompleteness of an initial scenario text is detected thanks to the semantic pattern instantiation. When the instantiation of a semantic pattern for a given clause of the initial scenario text is incomplete, L'Ecritoire detects the incompleteness and asks the user to complete the original statement.

For example, for the following scenario action *« a prompt for code is given »* the instantiated pattern *communication(give) [Agent : ? ; object : a prompt for code ; Source : ? ; Destination : ?]* shows missing elements. Every '?' above must be replaced by a term. This leads to the completed sentence: *« a prompt for code is given by the ATM to the user ».*

Scenario linguistic disambiguation. The syntactical analysis previous to pattern matching is used to detect anaphoric references in the initial scenario text and to ask the user to replace them by unambiguous terms. For example in the action « the user inserts his card in the ATM »: the anaphoric reference 'his' is detected; the user is asked to replace 'his' by a non-ambiguous term, in this case the 'customer', and the action is rephrased as « the user inserts the customer's card in the ATM ».

2.4 Discovery Rules

Discovery rules guide the L'Ecritoire user in discovering new goals and therefore, eliciting new requirement chunks. The discovery is based on the analysis of scenarios through one of the three proposed discovery strategies, namely the *refinement*, *composition* and *alternative* strategies. These strategies correspond to the three types of relationships among RCs introduced in section 2.1 above. Given a pair <G,Sc>:

- the composition strategy looks for goals Gi ANDed to G,
- the alternative strategy searches for goals Gj ORed to G,
- the refinement strategy aims at the discovery of goals Gk at a lower level of abstraction than G.

Composition (alternative) rules help in discovering ANDed (ORed) goals to G that are found at the same level of abstraction as G. The <G,Sc> chunk is processed by the refinement rules to produce goals at a lower level of abstraction than G. This is done by considering (in a similar way to that suggested by Cockburn [30]) each interaction in Sc as a goal. Thus as many goals are produced as there are interactions in Sc.

As shown in Figure 3, once a complete scenario has been authored, any of these three strategies can be followed. Thus, there is no imposed ordering on the flow of steps which instead, is dynamically defined.

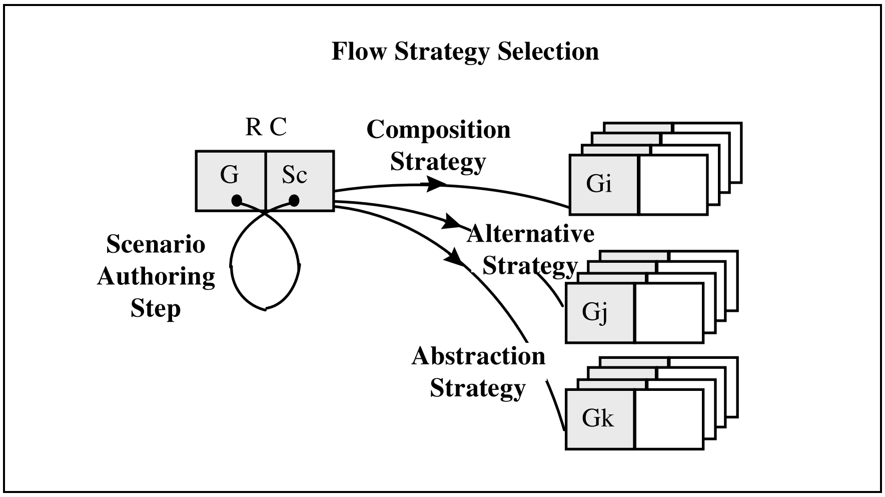


Fig. 3. Selecting a strategy

L'Ecritoire uses six discovery rules, two for each strategy. Rules can be applied at any of the three levels of abstraction, contextual, system interaction and system internal. A detail description of rules can be found in [38] [42]. As an example of a rule, we present the refinement rule R1 and exemplify it with the example of ATM system engineering.

Refinement guiding rule (R1)

Goal: Discover (from requirement chunk <G,Sc>)$_{So}$ (goals refined from G)$_{Res}$ (using every atomic action of Sc as a goal)$_{Man}$

Body:

Step1: Associate a goal Gi to every atomic action Ai in Sc. Gi refines G

Step2: Complement Gi by the manner 'in a normal way' which refers to the normal course of actions to attain the goal.

Step3: User evaluates the proposed panel of goals Gi and selects the goals of interest.

Step4: Requirement chunks corresponding to these selected goals are ANDed to one another

The guiding rule R1 aims at refining a given requirement chunk (*from RC<G,Sc>*)$_{So}$ by suggesting new goals at a lower level of abstraction than G (*goals refined from G*)$_{Res}$.

The refinement mechanism underlying the rule looks to every interaction between two agents in the scenario Sc as a goal for the lower level of abstraction (step1). Let us take as an example the scenario of the requirement chunk *RC < G, Sc>* presented below:

Goal G: Improve services to our customers by providing cash from the ATM

Scenario SC:

1- If the bank customer gets a card from the bank,

2- Then, the bank customer withdraws cash from the ATM

3- and the ATM reports cash transactions to the bank.

This scenario text corresponds to the structured textual form of the scenario as it results from the authoring step. The internal form is a set of semantic pattern instances which clearly identify three agents namely, the bank, the customer and the ATM as well as three interactions namely 'Get card', 'Withdraw cash' and 'Report cash transactions' corresponding to the three services involving the ATM. These services are proposed as goals of a finer grain than G, to be further made concrete by authoring scenarios for these goals.

We propose that these scenarios describe the normal course of actions. Thus, in the formulation of these goals according to the linguistic template (see section 2.1) the manner of every generated goal Gi is fixed to *'in a normal way'* (step2). This leads in the above example, to propose to the user the three following refined goals:

- *G 1: 'Get card from the bank in a normal way'*
- *G 2: 'Withdraw cash from ATM in a normal way'*
- *G 3: 'Report cash transactions to the bank in a normal way'*

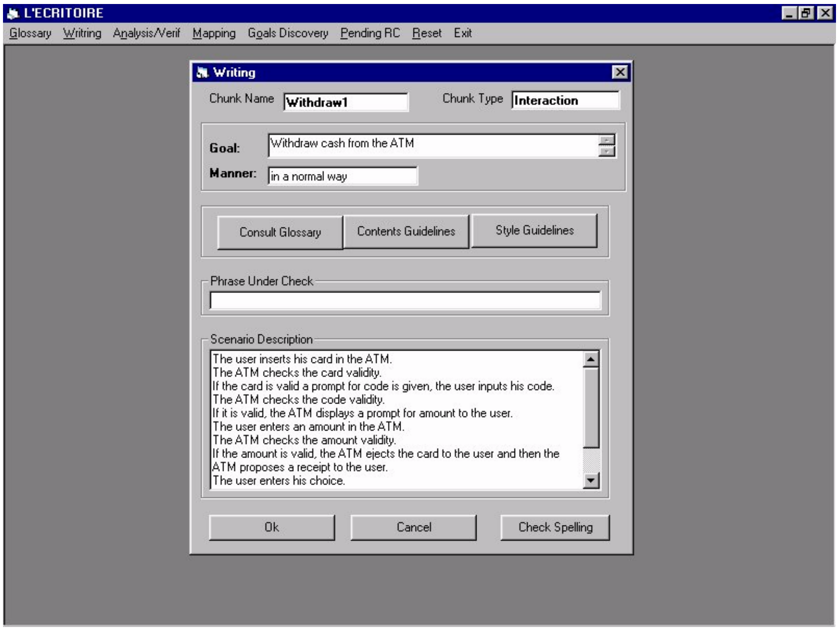
These three goals G_1, G_2, *and* G_3 are refined goals from G; they belong to the *interaction level* as G is at the *contextual level.* They are related to G by a *refined-by* relationship. Assuming that the user accepts the three suggested goals (step3), the corresponding requirement chunks RC_1, RC_2, RC_3 are introduced in the RC document and ANDed to one another (step4). These three RCs are at the interaction level.

3 The Usage Scenario

In this section, we illustrate the functionality of the L'Ecritoire tool through a usage scenario drawn from the ATM system. The scenario shows how the user of the tool interacts with L'Ecritoire to engineer system requirements. It illustrates a top down approach that, starting from a goal at the system interaction level and using the three discovery strategies of section 2, guides the elicitation of system requirements.

Step 1: Starting the Session

Let us start the session with the capture of a new RC. This requires two activities (a) entering the new goal and (b) authoring a scenario for this goal. Both these can be done in the window presented in Figure 6.



Chunk Name: Withdraw1
Chunk Type: Interaction
Goal: Withdraw cash from the ATM
Manner: in a normal way

The user inserts his card in the ATM.
The ATM checks the card validity.
If the card is valid a prompt for code is given, the user inputs his code.
The ATM checks the code validity.
If it is valid, the ATM displays a prompt for amount to the user.
The user enters an amount in the ATM.
The ATM checks the amount validity.
If the amount is valid, the ATM ejects the card to the user and then the ATM proposes a receipt to the user.
The user enters his choice.

Fig. 4. Capturing a new RC

(a) The user enters the goal *'Withdraw cash from the ATM in a normal way'*. The user has also to name the associated RC (Withdraw 1) and provide its type (interaction).

(b) A scenario is authored for the goal entered in (a) above. Since the scenario is entered in natural language, the user may use the facilities provided by the four buttons, Consult Glossary, Contents Guidelines, Style Guidelines, and Check Spelling. The first allows the user to consult the glossary to use the appropriate term. The second and the third provide contents and style guidelines appropriate for this type of RC. Finally, it is possible to perform a spell-check on the scenario being authored using the last button.

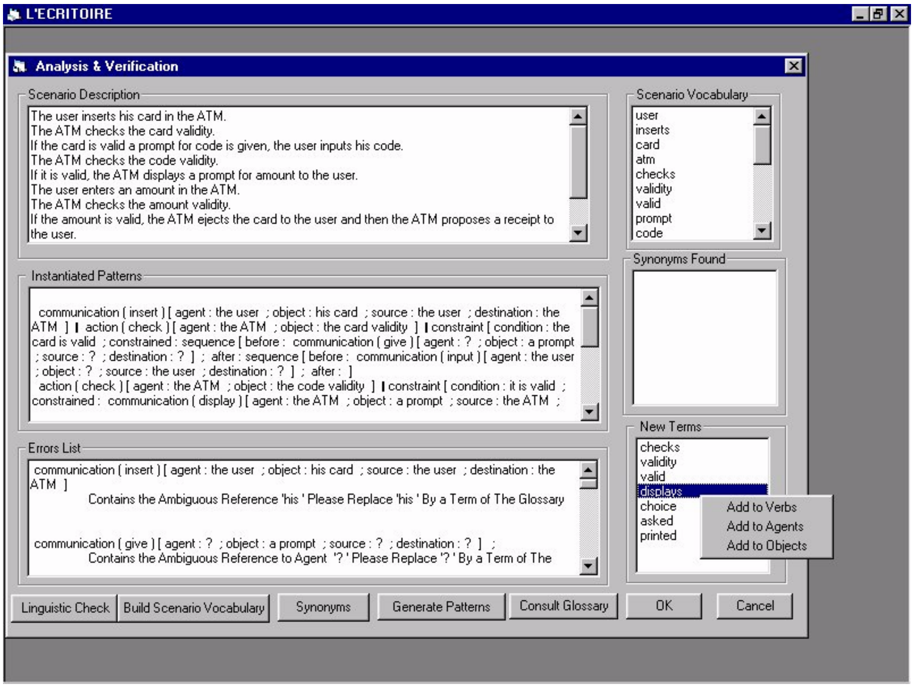
The authored scenario is reproduced below for ease of reference in the rest of this paper.

The user inserts his card in the ATM.
The ATM checks the card validity.
If the card is valid, a prompt for code is given, the user inputs his code.
The ATM checks the code validity.
If it is valid, the ATM displays a prompt for amount to the user.
The user enters an amount.
The ATM checks the amount validity.
If the amount is valid, the card is ejected and then the ATM proposes a receipt to the user.
The user enters his choice.
If a receipt was asked, the receipt is printed but before the ATM delivers the cash.

Step 2: Scenario Analysis

L'Ecritoire provides a guidance mechanism to progress in the process. This mechanism is activated by clicking the right button of the mouse. It gives a menu of rules that can be enacted upon the requirement chunk under consideration. Using this progress guidance mechanism, the user may decide to progress with the guided analysis of the scenario that was just captured. The Analysis and Verification window is shown in Figure 5. It appears with the textual scenario written during the previous step displayed in the text zone of the top left of the window.

To activate the linguistic devices, the user has to click the Generate Patterns button of Figure 5. The result of the linguistic analysis is shown as a set of instantiated patterns in the Instantiated Patterns widget. A list of errors found during pattern instantiation can be displayed in the Error List widget by clicking the button Linguistic Check. The errors reported in the Error List must be removed from the scenario. These corrections are performed in the Scenario Description widget and step 3 as described here is repeated till no errors are found. The corrected scenario is finally found in the Scenario Description widget. This scenario is stored in the L'Ecritoire repository.



```
The user inserts his card in the ATM.
The ATM checks the card validity.
If the card is valid a prompt for code is given, the user inputs his code.
The ATM checks the code validity.
If it is valid, the ATM displays a prompt for amount to the user.
The user enters an amount in the ATM.
The ATM checks the amount validity.
If the amount is valid, the ATM ejects the card to the user and then the ATM proposes a receipt to
the user.
```

Scenario Vocabulary: user, inserts, card, atm, checks, validity, valid, prompt, code

New Terms: checks, validity, valid, displays, choice, asked, printed

```
 communication ( insert ) [ agent : the user  ; object : his card  ; source : the user  ; destination : the
ATM  ] ❙ action ( check ) [ agent : the ATM  ; object : the card validity  ] ❙ constraint [ condition : the
card is valid  ; constrained : sequence [ before :  communication ( give ) [ agent : ?  ; object : a prompt
 ; source : ?  ; destination : ?  ]  ;  after : sequence [ before :  communication ( input ) [ agent : the user
 ; object : ?  ; source : the user  ; destination : ?  ]  ;  after :  ]
 action ( check ) [ agent : the ATM  ; object : the code validity  ] ❙ constraint [ condition : it is valid  ;
constrained :  communication ( display ) [ agent : the ATM  ; object : a prompt  ; source : the ATM  ;
```

```
 communication ( insert ) [ agent : the user  ; object : his card  ; source : the user  ; destination : the
ATM  ]
            Contains the Ambiguous Reference 'his ' Please Replace 'his ' By a Term of The Glossary


 communication ( give ) [ agent : ?  ; object : a prompt  ; source : ?  ; destination : ?  ]   ;
            Contains the Ambiguous Reference to Agent  '? ' Please Replace '? ' By a Term of The
```

Fig. 5. The analysis and verification window

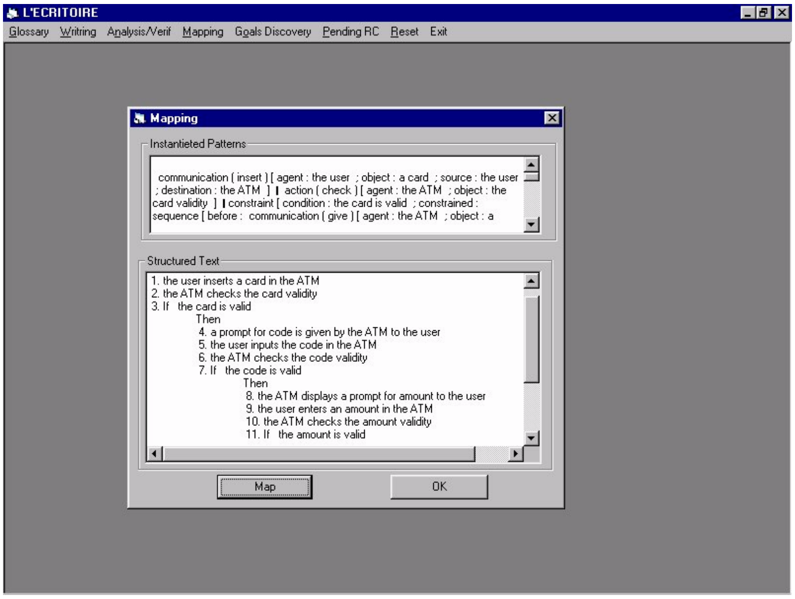
The corrected version of the initial scenario is shown below with the corrections in bold.

> The user inserts **a card in the ATM.**
> The ATM checks the card validity.
> If the card is valid a prompt for code is given **by the ATM to the user**, the user inputs the code **in the ATM**.
> The ATM checks the code validity.
> If **the code is valid**, the ATM displays **a prompt for amount to the user**.
> The user enters an amount **in the ATM**.
> The ATM checks the amount validity.
> If the amount is valid, **the ATM ejects the card to the user** and then the ATM proposes a receipt to the user.
> The user enters **the user's choice** in the ATM.
> If a receipt was asked the receipt is printed **by the ATM to the user** but before the ATM **delivers the cash to the user.**

Step 3: Structuring Scenario Text

The instantiated patterns corresponding to the text above represent the essence of the scenario as input by the user. Whereas these patterns can be used by L'Ecritoire internally, they are not in a readable form for the user. The purpose of step 4 is to map the patterns instances into a structured readable text.

Clicking the OK button of the previous window prompts the window of Figure 6 with its Instantiated Patterns widget containing the instantiated patterns. When the Map button is pressed then L'Ecritoire generates the structured text for the scenario and displays it in the Structured Text widget.


L'ECRITOIRE
Glossary Writing Analysis/Verif Mapping Goals Discovery Pending RC Reset Exit
Mapping
Instantieted Patterns
Structured Text
Map
OK

```
communication ( insert ) [ agent : the user  ; object : a card  ; source : the user
 ; destination : the ATM  ] ❙ action ( check ) [ agent : the ATM  ; object : the
card validity  ] ❙constraint [ condition : the card is valid  ; constrained :
sequence [ before :  communication ( give ) [ agent : the ATM  ; object : a
```

```
1. the user inserts a card in the ATM
2. the ATM checks the card validity
3. If   the card is valid
         Then
          4. a prompt for code is given by the ATM to the user
          5. the user inputs the code in the ATM
          6. the ATM checks the code validity
          7. If   the code is valid
                   Then
                    8. the ATM displays a prompt for amount to the user
                    9. the user enters an amount in the ATM
                    10. the ATM checks the amount validity
                    11. If   the amount is valid
```

Fig. 6. The Mapping window

The structured scenario is reproduced below for sake of clarity.

```
1. the user inserts a card in the ATM
2. the ATM checks the card validity
3. If  the card is valid
   Then
    4. a prompt for code is given by the ATM to the user
    5. the user inputs the code in the ATM
    6. the ATM checks the code validity
    7. If  the code is valid
          Then
           8. the ATM displays a prompt for amount to the user
           9. the user enters an amount in the ATM
           10. the ATM checks the amount validity
           11. If  the amount is valid
                 Then
                  12. the ATM ejects the card to the user
                  13. the ATM proposes a receipt to the user
                  14. the user input the user's choice
                  15. If  a receipt was asked
                        Then
                         16. the ATM delivers the cash to the user
                         17. the receipt is printed by the ATM to the user
```

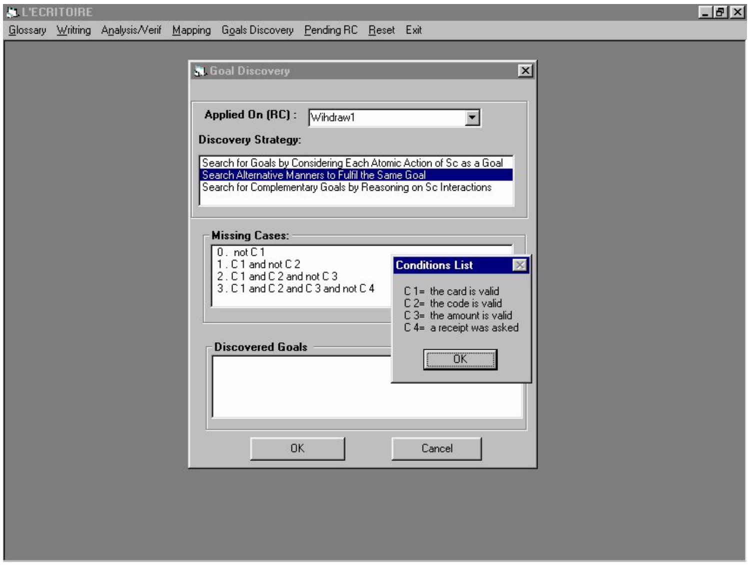
Step 4: Discover Goals from the Scenario

Upon using the progress guidance mechanism again, the user is presented with the three strategies, Refinement, *Complementary,* and *Alternative*, that can be deployed for goal discovery from the scenario. The user considers the properties of the scenario to decide the strategies to be adopted. Evidently, the scenario above has a large number of variants as shown by the number of If statements in it. Indeed, for every If statement at least one alternative way of proceeding can be described. Therefore the *Alternative strategy* must be deployed to guide in the discovery of those variants. Besides, an examination of the first three conditions of the scenario (lines 3, 7 and 11) shows that there are many ways in which these could be handled (for example a card could be invalid for different reasons). Therefore the user selects the *Refinement strategy* to help him/her eliciting internal requirements .

Having determined the two interesting strategies, the user decides to first select the Alternative strategy and then the Refinement one. The response of L'Ecritoire is shown in steps 5 and 6 respectively.

Step 5: Using the Alternative Strategy

In our illustration, the user selects the alternative strategy and within this strategy the rule to *Search alternative manners to fulfill the same goal.* The window of Figure 7 now appears on the screen with the Conditions List and Missing Cases widgets already filled in. The former contains the conditions of the If statements of the scenario


L'ECRITOIRE
Glossary Writring Analysis/Verif Mapping Goals Discovery Pending RC Reset Exit
Goal Discovery
OK
Cancel

Applied On (RC) : Wihdraw1

Discovery Strategy:

Search for Goals by Considering Each Atomic Action of Sc as a Goal
Search Alternative Manners to Fulfil the Same Goal
Search for Complementary Goals by Reasoning on Sc Interactions

Missing Cases:

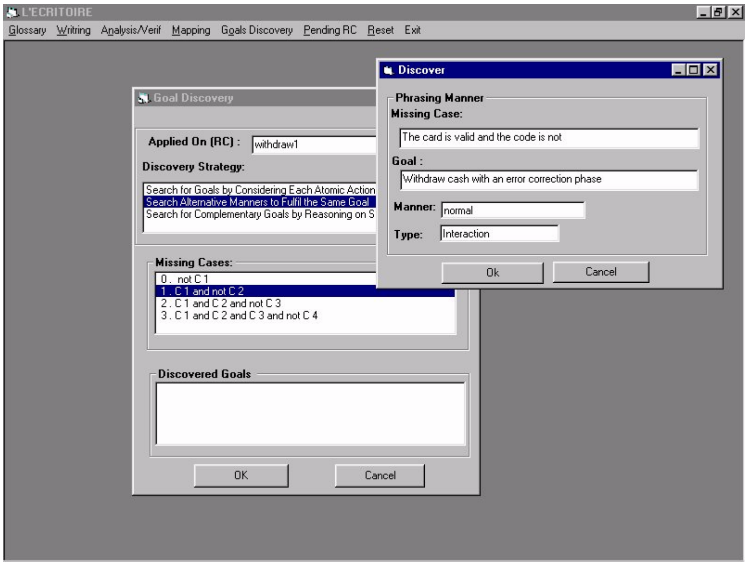
0 . not C 1
1 . C 1 and not C 2
2 . C 1 and C 2 and not C 3
3 . C 1 and C 2 and C 3 and not C 4

Discovered Goals

Conditions List

C 1= the card is valid
C 2= the code is valid
C 3= the amount is valid
C 4= a receipt was asked

Fig. 7. The goal discovery window for the alternative strategy


L'ECRITOIRE
Glossary Writring Analysis/Verif Mapping Goals Discovery Pending RC Reset Exit
Goal Discovery
OK
Cancel
Discover
Ok
Cancel

Applied On (RC) : withdraw1

Discovery Strategy:

Search for Goals by Considering Each Atomic Action
Search Alternative Manners to Fulfil the Same Goal
Search for Complementary Goals by Reasoning on S

Missing Cases:

0 . not C 1
1 . C 1 and not C 2
2 . C 1 and C 2 and not C 3
3 . C 1 and C 2 and C 3 and not C 4

Discovered Goals

Phrasing Manner

Missing Case: The card is valid and the code is not

Goal : Withdraw cash with an error correction phase

Manner: normal

Type: Interaction

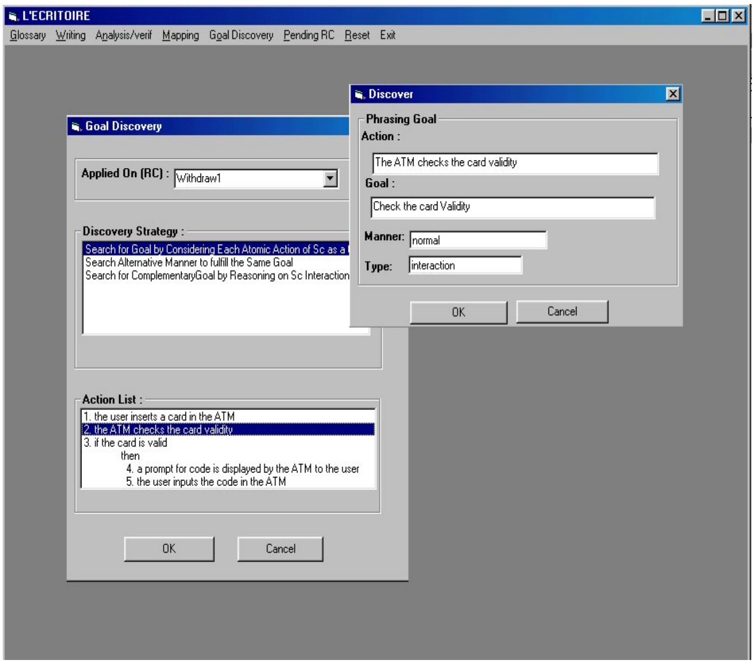
Fig. 8. The goal discovery window for the alternative strategy

of step 3. The ordering of the conditions reflects the nesting of the If statements in the scenario. This rule computes all possible combinations of the negation of these conditions and considers each of these as a missing case. It can be seen that l'Ecritoire performs a sort of completeness test here to discover missing cases. These cases are displayed in the Missing Cases Widget. As shown in Figure 7, there are four conditions C1 to C4 in the scenario. The rule computes the four missing cases, not C1, C1 and not C2, C1 and C2 and not C3, C1 and C2 and C3 and not C4.

The user now examines each of the candidate missing cases and if found relevant, formulates it as a goal. As soon as the user selects a case in the Missing Cases widget, the Discover window is prompted as shown in Figure 8. Thus the user can formulate the goal corresponding to this missing case. As illustrated in Figure 8 the case, C1 and not C2, is found relevant. For the ATM this means that it must react to a wrongly entered PIN code and so the user formulates the goal, *Withdraw cash with an error correction phase* (Figure 8).

Step 6: Using the Refinement Strategy

Let us now consider the second case, that of selection of the Refinement strategy, of step 4 above. The window shown in Figure 9 appears with the Action List widget already displayed. The list of actions corresponds to the atomic actions of the scenario which have been automatically extracted from the internal representation of the scenario stored in the L'Ecritoire repository. The user considers each action in this list as a candidate goal at the next lower abstraction level. Upon selection of such an action, the Discover window is prompted to allow rephrasing it as a goal. The type of the RC associated to this goal is automatically set to "internal". As shown in Figure 9, action 2 of the scenario of step 3 *"2. the ATM checks the card validity"* is refined as the goal *Check the card validity*. Similarly for action 6 the goal is *Check the code validity.*


L'ECRITOIRE
Glossary Writing Analysis/verif Mapping Goal Discovery Pending RC Reset Exit
Goal Discovery
Discover
OK
Cancel
OK
Cancel

Applied On (RC) : Withdraw1

Discovery Strategy :
Search for Goal by Considering Each Atomic Action of Sc as a
Search Alternative Manner to fulfill the Same Goal
Search for ComplementaryGoal by Reasoning on Sc Interaction

Action List :
1. the user inserts a card in the ATM
2. the ATM checks the card validity
3. if the card is valid
then
4. a prompt for code is displayed by the ATM to the user
5. the user inputs the code in the ATM

Phrasing Goal
Action : The ATM checks the card validity
Goal : Check the card Validity
Manner: normal
Type: interaction

Fig. 9. The goal discovery window for the refinement strategy

Step7: Terminating the Session

At this point the user has discovered three goals, namely, *Withdraw cash with an error correction phase, Check the card validity*, and *Check the code validity*. A scenario has still to be authored for each one of these from which new goals will be discovered. It can be done in this session itself or in another session. In the latter case, the current session is terminated. When a new session will be launched then L'Ecritoire will display the list of pending RCs.

4 Conclusion

Despite the importance of goals in the requirements elicitation process, experience shows difficulties in goal discovery. Vice versa, despite the concreteness of scenarios to capture some relevant aspects of the system To-Be, scenarios pose the coverage and completeness problems. One possible solution to mitigate the difficulties encountered in the separate use of goals or scenarios to discover accurate requirements is to combine them. The paper presents the l'Ecritoire approach that has been developed in this perspective.

The basis of the l'Ecritoire approach is the exploitation of the goal-scenario relationship but in the reverse direction. One can now talk of a tight coupling between goals and scenarios; in the forward direction this coupling promotes goal operationalisation whereas in the reverse direction it promotes goal discovery. Since, in the forward direction, scenarios represent a concrete, useful way of realising a goal, any technique which uses scenarios to discover goals shall produce potentially useful goals. This contributes to removing the fitness of use problem identified by Potts which leads to the generation of spurious, uninteresting or non-critical goals.

Since interactions expressed in scenarios are concrete and recognisable, the use of goal-scenario coupling for goal discovery helps in removing the 'fuzziness' that domain experts find in the notion of a goal. Instead, each interaction corresponds to goals. Again, goal discovery now becomes a natural process through interactions of scenarios and the goal-scenario coupling removes some of the mystery and ad-hocism associated with it. In this sense it helps in goal discovery.

Finally, the combination of composition, alternative and refinement rules alleviates the problem of goal reduction. By generating AND, OR relationships and goals at different abstraction levels, the approach automates and supports a major part of the requirements engineer's work.

References

1. Ross, D.T., Schoman, K.E.: Structured Analysis for Requirements Definition. IEEE Transactions on Software Engineering 3(1), 6–15 (1977)
2. Potts, C., Takahashi, K., Antòn, A.I.: Inquiry-based requirements analysis. IEEE Software 11(2), 21–32 (1994)
3. Rolland, C., Souveyet, C., Ben Achour, C.: Guiding goal modelling using scenarios. IEEE Transactions on Software Engineering, Special Issue on Scenario Management 24(12) (1998)

4. Rolland, C., Grosz, G., Kla, R.: Experience with goal-scenario coupling. in requirements engineering. In: Proceedings of the Fourth IEEE International Symposium on Requirements Engineering, Limerik, Ireland (1999)
5. van Lamsweerde, A.: From System Goals to Software Architecture. In: Bernardo, M., Inverardi, P. (eds.) SFM 2003. LNCS, vol. 2804, pp. 25–43. Springer, Heidelberg (2003)
6. Yu, E.: Modelling strategic relationships for process reengineering. Ph.D. Thesis, Dept. Computer Science, University of Toronto (1994)
7. Yue, K.: What does it mean to say that a specification is complete? In: Proc. IWSSD-4. Four International Workshop on Software Specification and Design, Monterrey, USA (1987)
8. Gote, O., Finkelstein, A.: Modelling the contribution structure underlying requirements. In: 1st Int. Workshop on Requirements Engineering: Foundation of Software Quality, Utrech, Netherlands (1994)
9. Ramesh, B., Powers, T., Stubbs, C., Edwards, M.: Implementing requirements traceability: a case study. In: Proceedings of the 2nd Symposium on Requirements Engineering (RE 1995), UK, pp. 89–95 (1995)
10. Ivankina, E., Salinesi, C.: An Approach to Guide Requirement Elicitation by Analysing the Causes and Consequences of Threats. In: 14th European - Japanese Conference on Information Modelling and Knowledge Bases, Skövde, Sweden (2004)
11. Lamsweerde, A.V., Darimont, R., Letier, E.: Managing conflicts in Goal-driven Requirements Engineering. IEEE Transactions on Software Engineering 24(11) (1998)
12. Lamsweerde, A.V., Letier, E.: Handling obstacles in goal-oriented requirements engineering. IEEE Transactions on Software Engineering, Special Issue on Exception Handling 26(10), 978–1005 (2000)
13. Lamsweerde, A.V., Dairmont, R., Massonet, P.: Goal directed elaboration of requirements for a meeting scheduler: Problems and Lessons Learnt. In: Proc. Of RE 1995 – 2nd Int. Symp. On Requirements Engineering, York, pp. 194–204 (1995)
14. Anton, A.I., Potts, C.: The use of goals to surface requirements for evolving systems. In: International Conference on Software Engineering (ICSE 1998), Kyoto, Japan, pp. 157–166, 19–25 (1998)
15. Haumer, P., Pohl, K., Weidenhaupt, K.: Requirements elicitation and validation with real world scenes. IEEE Transactions on Software Engineering, Special Issue on Scenario Management 24(12), 11036–1054 (1998)
16. Bubenko, J., Rolland, C., Loucopoulos, P., De Antonellis, V.: Facilitating ‘fuzzy to formal’ requirements modelling. In: IEEE 1st Conference on Requirements Enginering, ICRE 1994, pp. 154–158 (1994)
17. Anton, A.I.: Goal based requirements analysis. In: Proceedings of the 2nd International Conference on Requirements Engineering ICRE 1996, pp. 136–144 (1996)
18. Etien, A., Salinesi, C.: Managing Requirements in a co-evolution context. In: Proceedings of the 13th IEEE International Conference on Requirements Engineering (RE 2005), pp. 125–134 (2005)
19. Potts, C.: Fitness for use: the system quality that matters most. In: Proceedings of the Third International Workshop on Requirements Engineering: Foundations of Software Quality REFSQ 1997, Barcelona, pp. 15–28 (1997)
20. Caroll, J.M.: The Scenario Perspective on System Development. In: Carroll, J.M. (ed.) Scenario-Based Design: Envisioning Work and Technology in System Development (1995)
21. Mack, R.L.: Discussion: Scenarios as Engines of Design. In: Carroll, J.M. (ed.) Scenario-Based Design: Envisioning Work and Technology in System Development, pp. 361–387. John Wiley and Sons, Chichester (1995)
22. Potts, C., Takahashi, K., Anton, A.I.: Inquiry-based requirements analysis. IEEE Software 11(2), 21–32 (1994)

23. Cockburn, A.: Structuring Use Cases with Goals. Technical report. Human and Technology, 7691 Dell Rd, Salt Lake City, UT 84121, HaT.TR.95.1 (1995), `http://members.aol.com/acocburn/papers/usecases.htm`
24. Jarke, M., Tung Bui, X., Carroll, J.M.: Scenario management: an interdisciplinary approach. Requirements Engineering Journal 23(3-4), 155–173 (1998)
25. Anton, A.I., Mc Cracken, W.M., Potts, C.: Goal Decomposition and Scenario Analysis in Business Process Reengineering. In: Wijers, G., Wasserman, T., Brinkkemper, S. (eds.) CAiSE 1994. LNCS, vol. 811, pp. 94–104. Springer, Heidelberg (1994)
26. Holbrook, C.H.: A Scenario - Based Methodology for Conducting Requirements Elicitation. ACM SIGSOFT, Software Engineering Notes 15(1), 95–104 (1990)
27. Salinesi, C., Presso, M.J.: A method to analyse changes in the realisation of business intentions and strategies for information system adaptation. In: Proceedings of 6th International Enterprise Distributed Object Computing Conference (EDOC 2002), pp. 84–95 (2002)
28. Dano, B., Briand, H., Barbier, F.: A Use Case Driven Requirements Engineering Process. In: Third IEEE International Symposium On Requirements Engineering RE 1997, Antapolis, Maryland, IEEE Computer Society Press, Los Alamitos (1997)
29. Jacobson, I.: The Use Case Construct in Object-Oriented Software Engineering. In: Carroll, J.M. (ed.) Scenario-based design: envisioning work and technology in system development, pp. 309–336. John Wiley and Sons, Chichester (1995)
30. do Prado Leite, J.C.S., Rossi, G., Balaguer, F., Maiorana, A., Kaplan, G., Hadad, G., Oliveros, A.: Enhancing a Requirements Baseline with Scenarios. In: Third IEEE International Symposium On Requirements Engineering RE 1997, Antapolis, Maryland, pp. 44–53. IEEE Computer Society Press, Los Alamitos (1997)
31. Pohl, K., Haumer, P.: Modelling Contextual Information about Scenarios. In: Proceedings of the Third International Workshop on Requirements Engineering: Foundations of Software Quality REFSQ 1997, Barcelona, pp. 187–204 (1997)
32. CREWS Team, The CREWS glossary, CREWS report 98-1 (1998), `http://SUNSITE.informatik.rwth-aachen.de/CREWS/reports.htm`
33. Fillmore, C.: The case for case. In: Holt, Rinehart, Winston (eds.) Universals in linguistic theory, pp. 1–90. Bach & Harms Publishing Company (1968)
34. Dik, S.C.: The theory of functional grammar, part I: the structure of the clause. Functional Grammar Series. Fories Publications (1989)
35. Schanck, R.C.: Identification of conceptualisations underlying natural language. In: Shanck, R.C., Colby, K.M. (eds.) Computer models of thought and language, pp. 187–247. Freeman, San Francisco (1973)
36. Prat, N.: Goal formalisation and classification for requirements engineering. In: Proceedings of the Third International Workshop on Requirements Engineering: Foundations of Software Quality REFSQ 1997, Barcelona, pp. 145–156 (1997)
37. Ben Achour, C.: Requirements Extraction From Textual Scenarios. PhD Thesis, University Paris 6 Jussieu (1999)
38. Rolland, C., Ben Achour, C.: Guiding the construction of textual use case specifications. Data & Knowledge Engineering Journal 25(1), 125–160 (1997)
39. Salinesi, C.: Authoring Use Cases. In: Alexander, I., Maiden, N. (eds.) Scenarios & Use Cases, Stories through the System Life-Cycle, John Wiley and Sons, Chichester (2004)
40. Chomsky, N.: Structures Syntaxiques. Editions du Seuil, Paris (1969)
41. Ben Achour, C., Rolland, C., Maiden, N.A.M., Souveyet, C.: Guiding Use Case authoring: results of an empirical study. In: IEEE International Symposium on Requirements Engineering (RE 1999), Essen Germany (1999)
42. Tawbi, M.: Crews-L'Ecritoire: un guidage outillé du processus d'Ingénierie des Besoins. Ph.D. Thesis University of Paris 1 (2001)
43. Mylopoulos, J., Chung, L., Nixon, B.: Representing and using nonfunctional requirements: aprocess-oriented approach. IEEE Transactions on Software Engineering 18(6), 483–497 (1992)

44. Yu, E., Mylopoulos, J.: Understanding « Why » in Software Process Modelling, Analysis and Design. In: 16th Int. Conf. Software Engineering, Sorrento, Italy (1994)
45. Yu, E., Mylopoulos, J.J.: Using goals, rules, and methods to support reasoning in business process reengineering. International Journal of Intelligent Systems in Accounting, Finance and Management 5(1), 1–13 (1996)
46. Mylopoulos, J., Chung, L., Yu, E.: From Object Oriented to Agent Oriented Requirementes Analsyis. Communications of the ACM 42(1), 31–37 (1999)
47. Fuxman, A., Pistore, M., Mylopoulos, J., Traverso, P.: Model checking early requirements specification in Tropos. In: Proc. of the 5th Int. Symposium on Requirements Engineering (RE 2001), Toronto, Canada (2001)
48. Giorgini, P., Mylopoulos, J., Nicchiarelli, E., Sebastiani, R.: Reasoning with Goal Models. In: Spaccapietra, S., March, S.T., Kambayashi, Y. (eds.) ER 2002. LNCS, vol. 2503, pp. 167–181. Springer, Heidelberg (2002)
49. Liu, L., Yu, E., Mylopoulos, J.: Security and Privacy Requirements Analysis within a Social Setting. In: 11th IEEE International Requirements Engineering Conference (RE 2003), Monterey Bay, California, USA, September 2003, p. 151 (2003)
50. Castro, J., Silva, C., Mylopoulos, J.: Modeling Organizational Architecutreal Styles in UML. In: Eder, J., Missikoff, M. (eds.) CAiSE 2003. LNCS, vol. 2681, pp. 11–126. Springer, Heidelberg (2003)
51. Yu, Y., Sampaio de Leite, J., Mylopoulos, J.: From Goals to Aspects: Discovering Aspects from Requirements Goal Models. In: 12th International Requirements Engineering Conference (RE 2004), Tokyo, Japan, pp. 38–47 (2004)
52. Yu, Y., Niu, N., Gonzalez Baixauli, B., Candillon, W., Mylopoulos, J., Easterbrook, S., Sampaio de Leite, J., VenWormhoubt, G.: Tracing and Validating Goal Aspects. In: 15th IEEE International Requirements Engineering Conference (RE 2008), Barcelona, Spain (2008)
53. Liu, L., Yu, E.: Designing Information Systems in Social Context: A Goal and Scenario Modelling Approach. In: Persson, A., Stirna, J. (eds.) CAiSE 2004. LNCS, vol. 3084, pp. 187–203. Springer, Heidelberg (2004)

Enhancing Tropos with Commitments
A Business Metamodel and Methodology

Pankaj R. Telang and Munindar P. Singh

Department of Computer Science
North Carolina State University
Raleigh, NC 27695-8206, USA
prtelang@ncsu.edu, singh@ncsu.edu

Abstract. This paper motivates a novel metamodel and methodology for specifying cross-organizational business interactions that is based on Tropos. Current approaches for business modeling are either high-level and semiformal or formal but low-level. Thus they fail to support flexible but rigorous modeling and enactment of business processes. This paper begins from the well-known Tropos approach and enhances it with commitments. It proposes a natural metamodel based on commitments and a methodology for specifying a business model. This paper includes an insurance industry case study that several researchers have previously used.

1 Introduction

Modern organizations form complex business relationships with their partners to create and provide value to their customers. Due to competitive pressures, modern organizations are continually forced to improve their operations. Such improvements increasingly involve outsourcing noncore business tasks, and other redistributions and realignments of business functions. A business model serves as a starting point for realizing the IT systems required to support the operations of these organizations.

Current approaches are inadequate for capturing business models in a manner that is both flexible and formal. On the one hand, management scientists have proposed a number of high-level business metamodels. However, these models are semiformal, and are useful primarily for valuation and profitability analysis. On the other hand, computer scientists have proposed low-level business metamodels, which consider abstractions centered on data and control flow. These approaches fail to capture the business meaning of the interactions.

This paper motivates a novel business metamodel and methodology based on Tropos [2], an established agent-oriented engineering methodology, as enhanced with commitments. Tropos provides a well-defined requirements engineering approach for modeling agents, and their mutual dependencies. However, it lacks appropriate treatment of agent commitments. Commitments [6] are a well-studied concept for modeling business interactions. Commitments help capture the business meaning of interactions in a manner that supports judgments of compliance while enabling flexible enactment. Our proposed metamodel benefits from the goal, task, and dependency modeling offered by Tropos, and the semantics and flexibility offered by commitments. We exercise our approach on an insurance industry scenario, a well-known case study from the literature.

Contributions. The main contribution of this paper is a novel agent-oriented business metamodel and methodology. A real-life insurance claim processing scenario validates our proposal.

Organization. Section 2 presents our business metamodel and methodology. Section 3 applies the methodology to create a business model for an insurance claim processing scenario. Section 4 introduces the notion of agent conformance and presents an approach for verifying it. Section 5 compares our approach with related work.

2 Metamodel and Methodology

In our treatment, a business model captures the business organizations involved in conducting a specified element of business and the commitments between them. An organization executes business tasks either to achieve its own goals, or to satisfy its commitments toward another organization. We now define the concepts used by our metamodel using which such models may be expressed. With the exception of commitment, these concepts are based on Tropos.

Agent: A computational representation of a business organization. An agent abstraction captures the autonomy and heterogeneity of a real-world business. An agent has goals and capabilities to execute tasks. It enters into business relationships with other agents to acquire capabilities, that is, to get other agents to execute tasks on its behalf.

Role: An abstraction over agents via their related sets of commitments and tasks. An agent can adopt one or more roles in a business model.

Goal: A state or condition of the world that an agent would like to bring about or have brought about. In simple terms, a goal is an end. AND-OR decomposition helps decompose a root goal into subgoals.

Task: An abstract way of executing a business activity, that is, the means by which a goal can be achieved. Similar to a goal, a root task can be decomposed into subtasks.

Dependency: A relationship between two roles where one role depends upon the other for achieving a goal or executing a task. The former role is called the *depender*, the latter is called the *dependee*, and the object of the dependency is called the *dependum*.

Commitment: A commitment $\mathsf{C}(\text{R1}, \text{R2}, \mathsf{p}, \mathsf{q})$ denotes that the role R1 is responsible to the role R2 for bringing about the the condition q if p holds. In this commitment, R1 is the *debtor*, R2 is the *creditor*, p is the *antecedent*, and q is the *consequent*. A commitment may be *created*. When its condition is brought about, it is *discharged*. The creditor may *assign* a commitment to another agent. Conversely, a debtor may *delegate* a commitment to another agent. A debtor may also *cancel* a commitment and a creditor may *release* the debtor from the commitment. The above operations describe how a commitment can be manipulated.

A traditional obligation specifies a condition that an agent ought to bring about. Unlike commitments, an obligation cannot be assigned, delegated, canceled, or released.

Table 1 outlines the steps in our proposed methodology. The subsections below describe each step in detail.

Table 1. Methodology steps

Step	Description	Input	Output
S1	Identify agents and roles	Scenario description, process flows, domain knowledge	Agents and roles
S2	Determine goals and goal dependencies	Roles, scenario description, process flows, domain knowledge	Goals and goal dependencies
S3	Identify tasks and task dependencies	Roles, goal dependencies, scenario description, process flows, domain knowledge	Tasks and task dependencies
S4	Identify commitments	Task dependencies, scenario description, process flows, domain knowledge	Commitments

2.1 Step S1: Agent and Role Identification

Agents represent the business organizations participating in the scenario of interest. A scenario description typically specifies the agents using terms like company, partner, and organization. If there is a single agent of its kind, then the scenario description usually specifies a unique name for it. In case we have multiple agents of the same kind, a scenario description may specify a role name. For each uniquely named agent, the business functions it provides yields the associated role.

A traditional process flow, if available, can help us identify the roles. For example, a *role* in a choreography corresponds to a role in our business model. Similarly, *partner link* in an orchestration, corresponds to a role in our model.

2.2 Step S2: Determine Goals and Dependencies

This step iteratively determines the goal dependencies between the roles. First, it identifies the main roles and their high-level goal dependencies. Second, using means-end and AND-OR decomposition analysis, it refines the high-level root goals into subgoals. Third, this step introduces roles that adopt these subgoals. It then iteratively refines the subgoals until no new dependencies arise.

The scenario description may explicitly or implicitly specify business dependencies between participating organizations. These dependencies yield the goal dependencies between the roles. In cases where the dependencies are not clear from the scenario, additional business insight may be needed.

Thus, Step S2 identifies goals adopted by different roles, that is, the goals of each dependee role.

2.3 Step S3: Determine Tasks and Dependencies

For each role, this step refines the goal dependencies from Step S2 into task dependencies. A set of tasks achieves a goal. The means-end analysis identifies the set of tasks required for achieving a goal. A task is refined into subtasks using AND-OR decomposition. The refined tasks identify dependencies that are not evident at the higher level task. The decomposition iterates until no new task dependencies emerge.

As we analyze additional roles, we may discover new task dependencies, requiring the addition of any missing tasks and goals to the (potentially already analyzed) roles.

2.4 Step S4: Identify Commitments

This step identifies commitments between roles in terms of tasks. It analyzes each task dependency from Step S3 to identify if a commitment exists corresponding to that dependency. For a task dependency, if the dependee is obliged to the depender for executing the dependum task, then a commitment exists, where the depender is the creditor, the dependee is the debtor, and the dependum task is the consequent. The antecedent of the commitment is determined by identifying the tasks that the debtor requires as prerequisites for executing the consequent task. If the dependee is not obliged to the depender for executing a task, then no commitment exists. This implies that the dependee executes the task to achieve its internal goal.

Although the scenario description and process flow may contain information that yields commitments, additional human insight is typically required to correctly identify the commitments.

3 Methodology Applied to a Real-World Case

This section describes a real-world insurance claim processing use case. It further describes an application of our methodology and the resulting model for that use case. Figure 1 clarifies the notation used in the figures that follow.

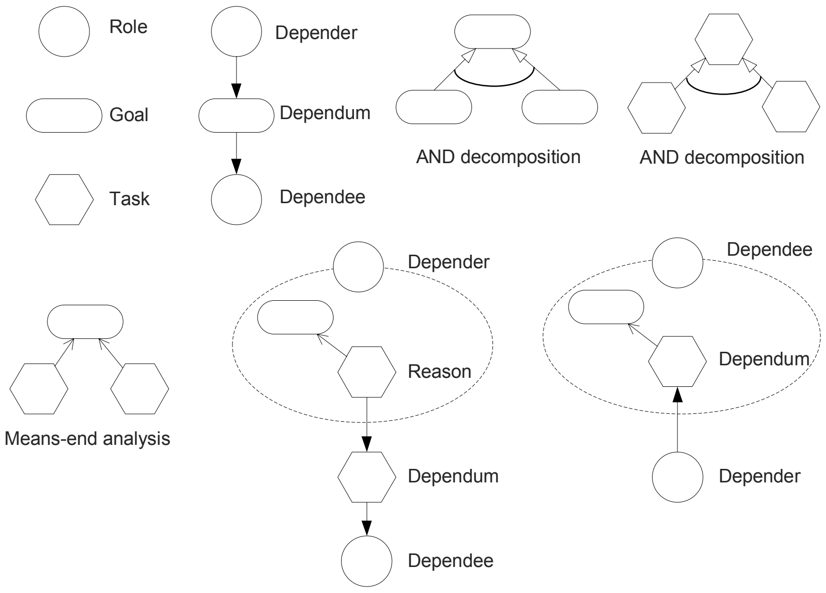


Fig. 1. Notations

3.1 Insurance Claim Processing Scenario

AGFIL [3] is an insurance company in Ireland. It underwrites insurance policies and covers losses incurred by policy holders. AGFIL provides an emergency service to its policy holders. Figure 2 shows the parties involved in providing emergency service, their individual processes, and the control and data flows among these processes.

To provide emergency service, AGFIL must provide claim reception and vehicle repair to the policy holders. Additionally, it needs to assess claims to protect itself against fraud. AGFIL uses its partners, Europ Assist, Lee Consulting Services (CS), and various repairers, for executing these tasks. Europ Assist provides a 24-hour help-line to customers for reporting a claim, and provides them the name of an approved repairer facility. Lee CS performs the necessary assessment and additionally presents invoices to AGFIL on behalf of the repairers. Several approved repairers provide repair services. AGFIL remains in charge of making the final decision on claim approvals, and making the payment.

3.2 Step S1

The insurance claim processing scenario from Section 1 specifies AGFIL, EA, and Lee CS as uniquely named agents. The process flow also shows these agents. These agents serve the business functions of insurer, call center, and assessor, respectively. Therefore, we can induce the corresponding roles from them. The description additionally specifies the repairer, claim adjustor, and policy holder roles that are enacted by multiple agents.

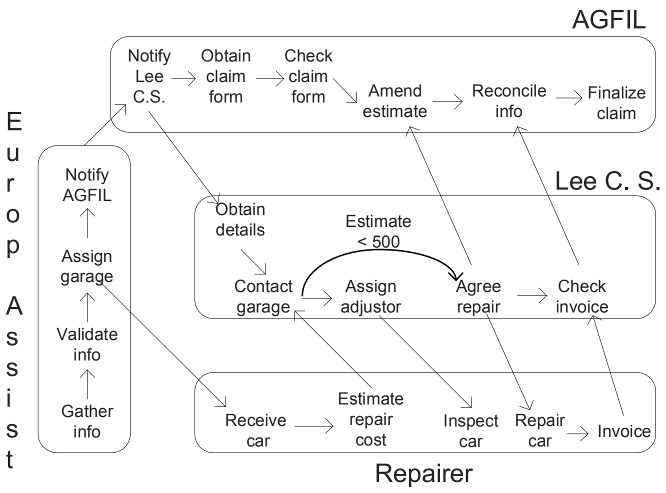


Fig. 2. Insurance claim processing [3]

3.3 Step S2

In the AGFIL scenario, the main roles are insurer and policy holder. A policy holder depends upon the insurer for receiving emergency service and, in exchange, the insurer depends upon the policy holder for the insurance premium payment. Fig. 3 shows these dependencies using the Tropos notation.

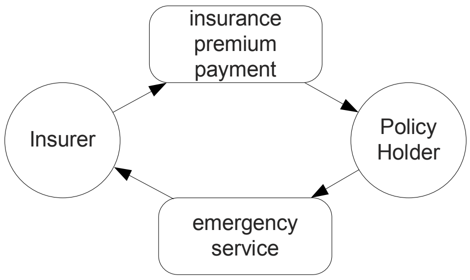


Fig. 3. Insurer and policy holder goal dependencies

Using AND decomposition, in Fig. 4, the insurer's goal of emergency service yields subgoals of claim reception, claim assessment, vehicle repair, and claim finalization. Among these, the policy holder depends on the insurer for vehicle repair and claim reception. The insurer requires additional subgoals to provide emergency service but these additional goals do not involve a dependency from the policy holder. As the goal structure is refined and later, in Step S3, when the task structure is identified, a dependency between one of these subgoals and the policy holder may be discovered.

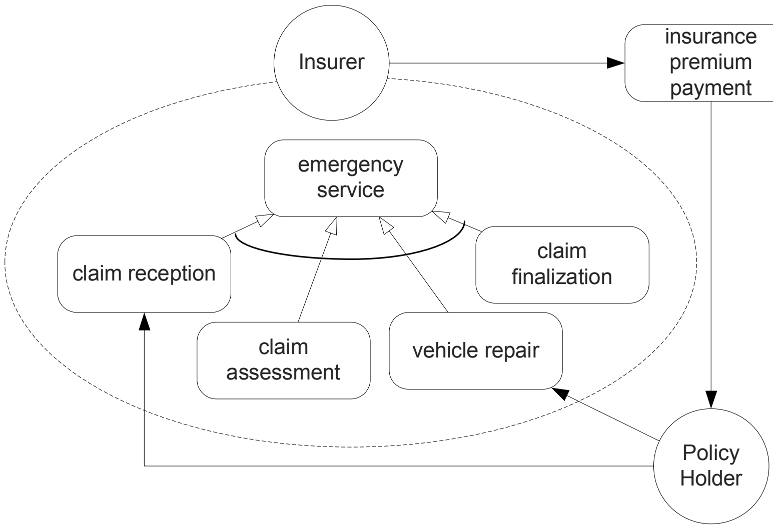


Fig. 4. Emergency service goal decomposition

In Fig. 5, the insurer delegates claim reception to the call center. The policy holder now depends upon the call center for claim reception. In exchange for claim reception, the insurer pays service charges to the call center. Fig. 5 omits the claim assessment and finalization subgoals as they are not dependent upon other roles.

In Fig. 6, the insurer delegates claim assessment to the assessor. The assessor role and a dependency from the insurer to the assessor is added to the model. In exchange, the assessor depends upon the insurer for the payment of assessment fees.

Fig. 7 shows all roles and goal dependencies. The assessor delegates vehicle inspection, which is a subgoal of claim assessment, to an adjustor. The adjustor,

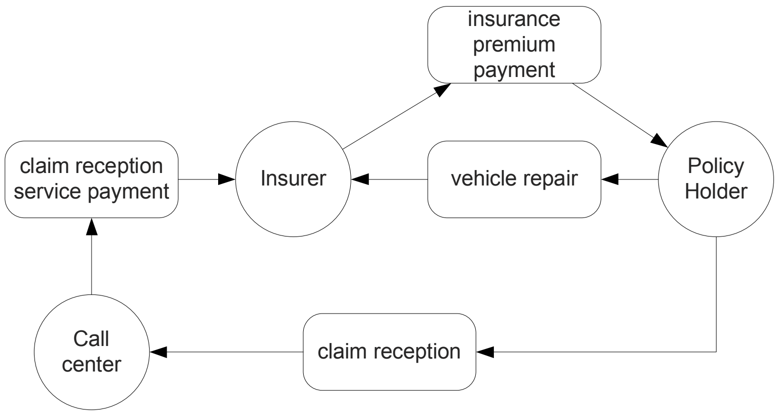


Fig. 5. Insurer delegates claim reception

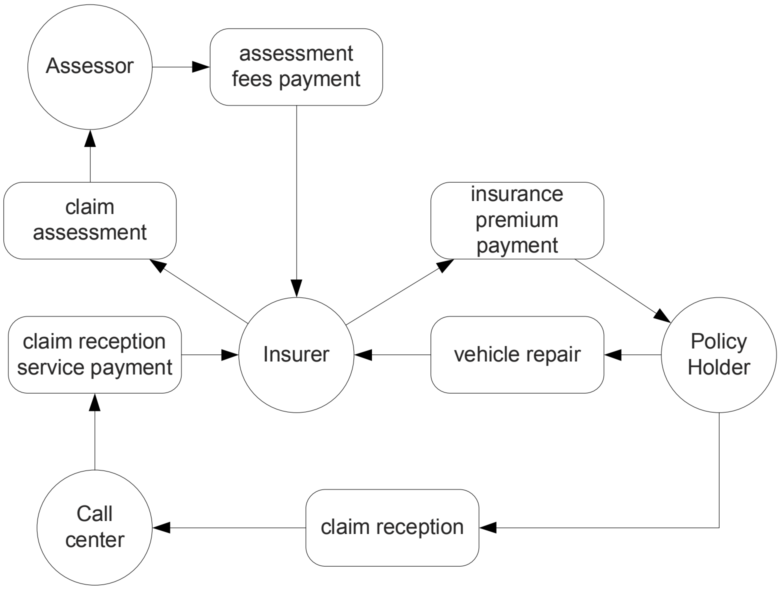


Fig. 6. Insurer delegates claim assessment

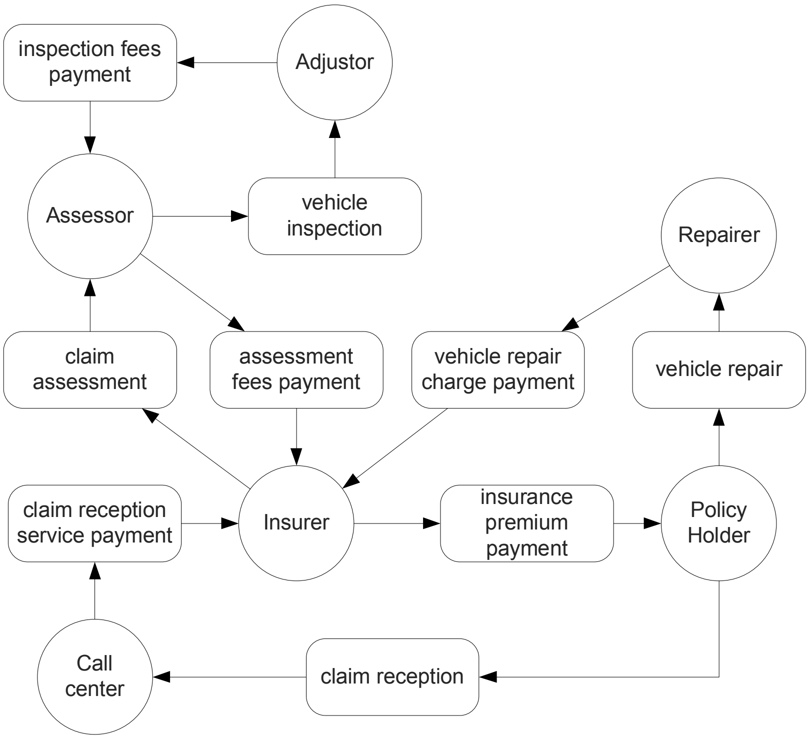


Fig. 7. AGFIL goal dependency model

in exchange, depends upon the assessor for inspection fees payment. The insurer delegates vehicle repair to a repairer. In exchange, the repairer depends upon the insurer for vehicle repair payment.

3.4 Step S3

Fig. 8 shows the tasks and the task dependencies we identify by analyzing the call center role. From S2, the call center has the goal of claim reception. By performing means-end analysis on this goal, we obtain tasks of gathering claim information, assigning garage, sending claim, and validating claim.

When the policy holder reports an accident, the call center gathers claim information and assigns a garage. This means that the policy holder depends upon the call center for gathering claim information and assigning a garage. Additionally, the repairer depends on the call center to assign a garage.

Using AND-OR decomposition of the validate claim information task, we obtain two subtasks: request policy information and validate. The call center depends upon the insurer for providing policy information, and it performs validation without any dependency.

From Step S2, the call center depends on the insurer for payment of the claim reception charge. This yields a dependency from the call center to the insurer for the task of paying claim reception charge. The call center executes a task of receiving this payment and it derives a new call center goal of service charge collection.

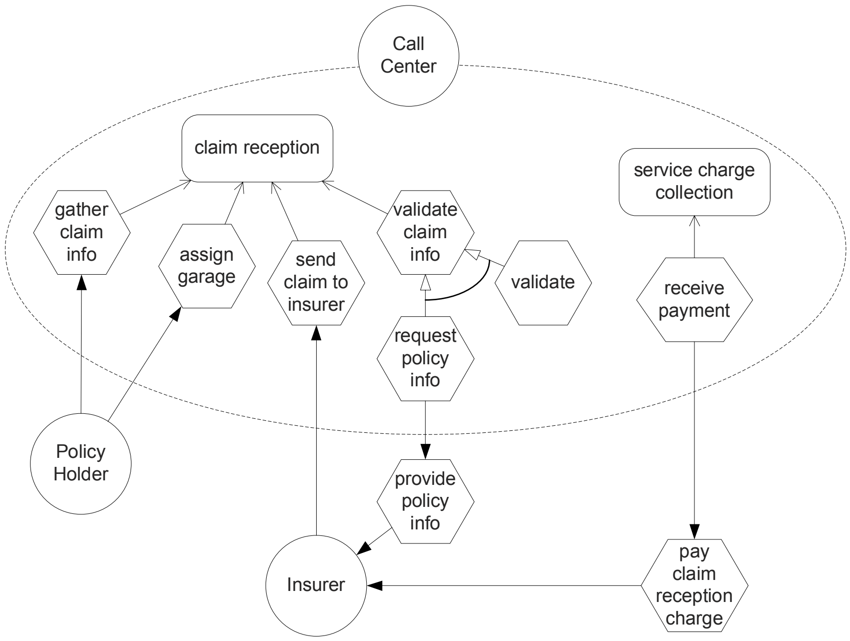


Fig. 8. Call center task dependencies

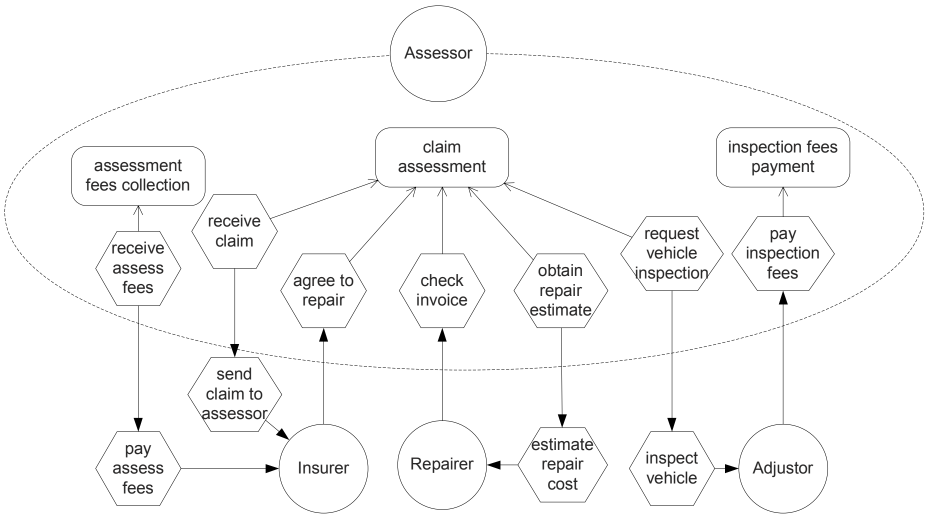


Fig. 9. Assessor task dependencies

The call center sends a validated claim to the insurer for further processing. This yields a dependency from the insurer to the call center for sending the claim.

Fig. 9 shows the tasks and the task dependencies derived for the assessor role. The assessor has goals of claim assessment and inspection fees payment. A new goal of assessment fees collection is derived similarly to the goal of receiving service payment of the call center.

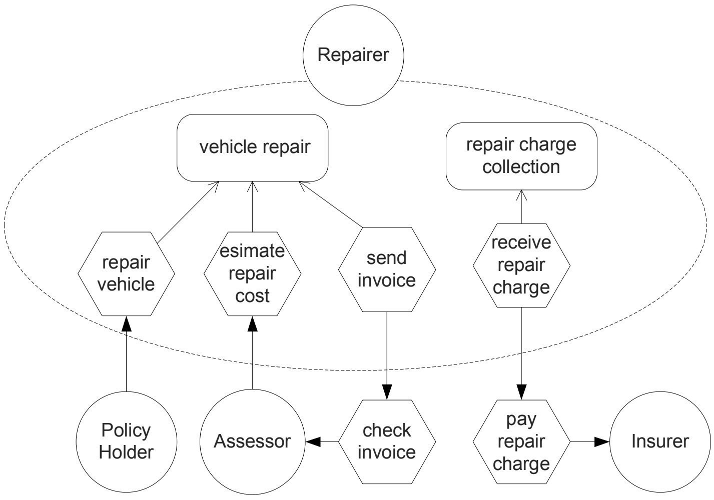


Fig. 10. Repairer task dependencies

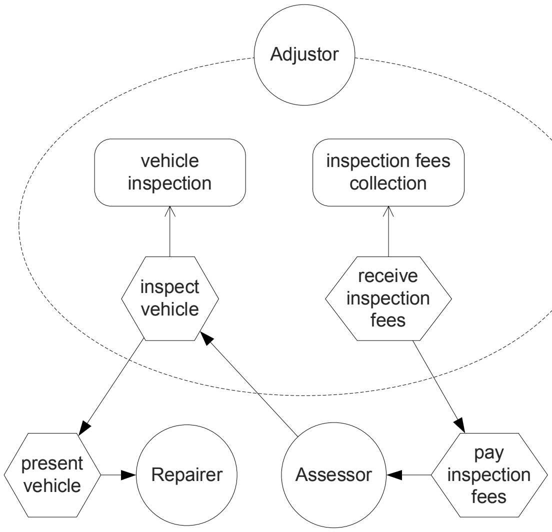


Fig. 11. Adjustor task dependencies

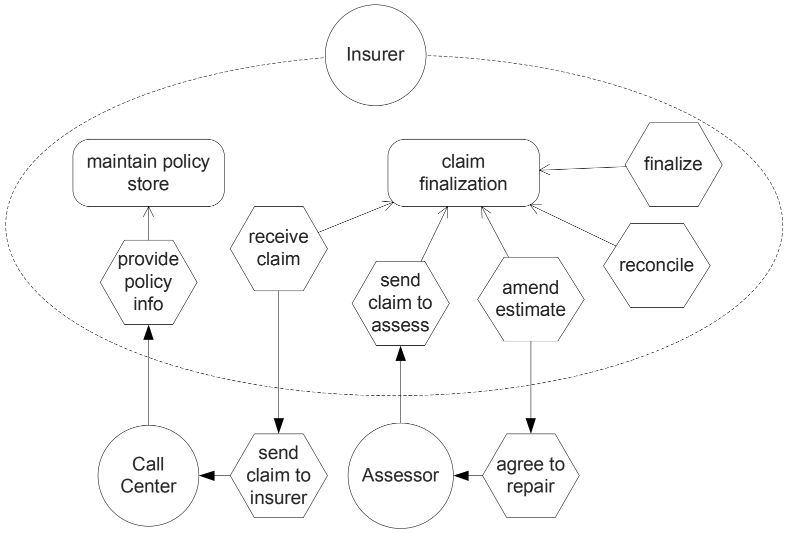


Fig. 12. Insurer task dependencies: Business function goals

The goal of assessment fees collection requires a receive assessment fees task, which depends upon the insurer's pay assessment fees task. The tasks needed to cover claim assessment goal are receive claim, check invoice, agree to repair, obtain repair estimate, and inspect vehicle. The receive claim task depends upon the insurer's task of sending claim. The insurer depends upon the check invoice and agree to repair tasks performed by the assessor. The repairer depends upon

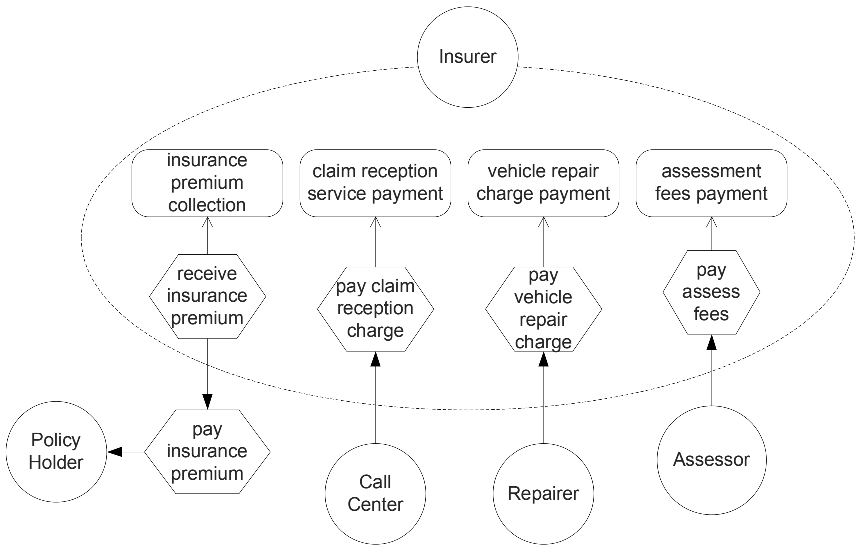

Insurer
insurance premium collection
claim reception service payment
vehicle repair charge payment
assessment fees payment
receive insurance premium
pay claim reception charge
pay vehicle repair charge
pay assess fees
Policy Holder
pay insurance premium
Call Center
Repairer
Assessor

Fig. 13. Insurer task dependencies: Payment related goals

the assessor for checking the invoice and agreeing to repair. For obtaining the repair estimate, the assessor depends upon the repairer to estimate the repair cost. The assessor depends upon the adjustor to inspect a vehicle. The goal of inspection fees payment requires a task of paying inspection fees to the adjustor.

Similarly, we analyze repairer, adjustor, insurer, and policy holder roles to obtain the task dependencies shown in Figs. 10, 11, 12, 13, and 14 respectively.

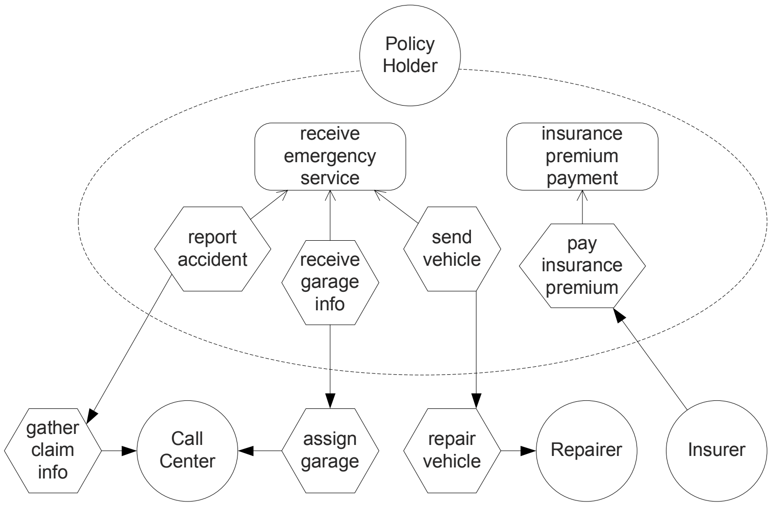


Fig. 14. Policy holder task dependencies

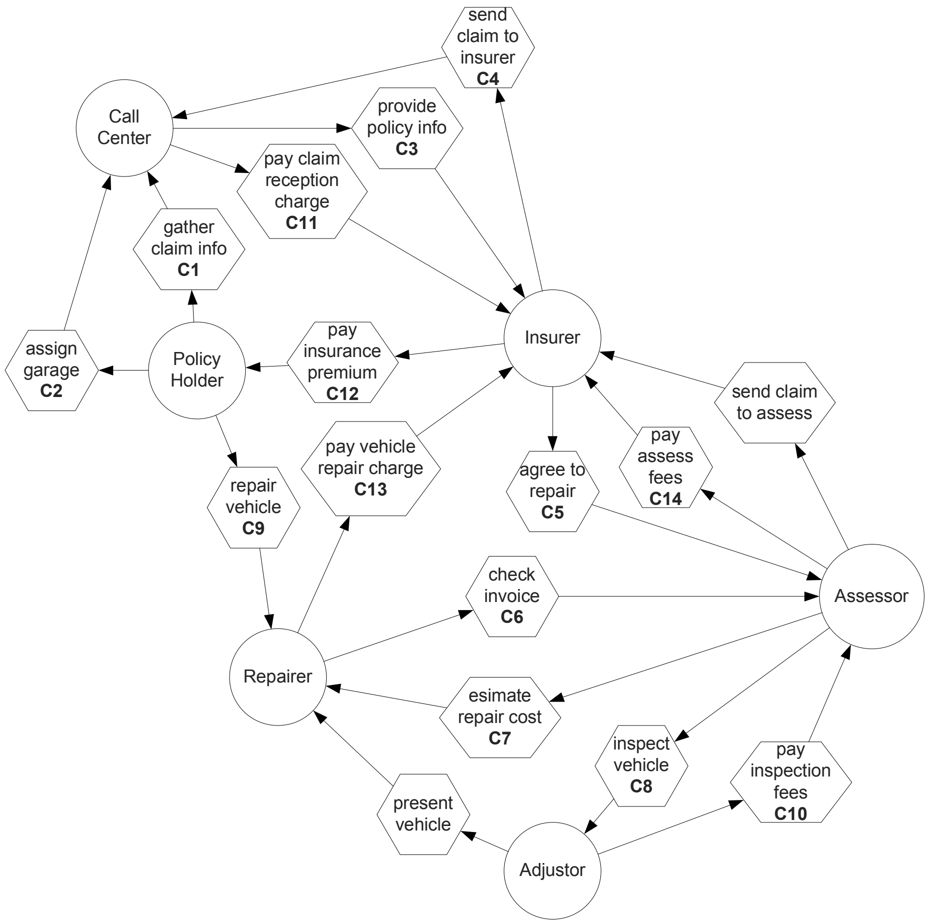


Fig. 15. All task dependencies

Fig. 15 consolidates the task dependencies among all roles. This figure shows only the dependum tasks and hides the tasks that are reasons for the individual dependencies. For example, Fig. 15 shows the pay repair charge task but does not show the receive repair charge task.

3.5 Step S4

This step identifies the commitments for each task dependency from Step S3. Fig. 15 annotates each task dependency with the corresponding commitment. Table 2 summarizes these commitments.

The commitment C1 means that the call center commits to the policy holder for gathering claim information if the policy holder reports an accident. In C2, the call center commits to assigning a garage if the policy holder reports an accident and if the claim request is valid. In C3, the insurer commits to providing

Table 2. Commitments for AGFIL scenario

Id	Task	Commitment
1	gather claim info	C(Call center, Policy holder, reportAccident, gatherInfo)
2	assign garage	C(Call center, Policy holder, reportAccident $\wedge$ validClaim, assignGarage)
3	provide policy info	C(Insurer, Call center, reqPolicyInfo, providePolicyInfo)
4	send claim to insurer	C(Call center, Insurer, reportAccident $\wedge$ validClaim $\wedge$ payClaimRecCharge, sendClaimToInsurer)
5	agree to repair	C(Assessor, Insurer, sendClaimToAssess $\wedge$ payAssessFees, agreeToRepair)
6	check invoice	C(Assessor, Repairer, sendInvoice, checkInvoice)
7	estimate repair cost	C(Repairer, Assessor, reqEstimate, estimateRepairCost)
8	inspect vehicle	C(Adjustor, Assessor, reqInspection $\wedge$ payInsFees, inspectVehicle)
9	repair vehicle	C(Repairer, Policy holder, validClaim, repairVehicle)
10	pay inspection fees	C(Assessor, Adjustor, inspectVehicle, payInsFees)
11	pay claim reception charge	C(Insurer, Call center, C1 $\wedge$ C2 $\wedge$ C4, payClaimRecCharge)
12	pay insurance premium	C(Policy holder, Insurer, C9, payInsurancePremium)
13	pay vehicle repair charge	C(Insurer, Repairer, repairVehicle, payRepairCharge)
14	pay assessment fees	C(Insurer, Assessor, agreeToRepair, payAssessFees)

policy information if the call center requests it. The commitment C4 means that the call center commits to sending a claim to the insurer if it receives a valid claim and the insurer pays claim reception charge. In C5, the assessor commits to the insurer to negotiate and bring about the agreement to repair provided the insurer requests claim assessment and pays the assessment fees. Commitment C6 means that the assessor commits to the repairer for checking the invoice and for forwarding it to the insurer, provided the repairer sends the invoice. In C7, the repairer commits to the assessor for estimating the repair cost if requested. In C8, the adjustor commits to the assessor for inspecting the vehicle if the assessor requests inspection, and pays for it. The commitment C9 means that the repairer commits to the policy holder for repairing the vehicle provided the claim is valid. In C10, the assessor commits to paying the adjustor for inspection if the vehicle is inspected. The commitment C11 means that the insurer commits to the call center for paying if the call center creates commitments C1, C2, and

C4. That is, if the call center commits to gathering claim information, assigning a garage, and sending the claim to insurer. In C12, the policy holder commits to paying the insurance premium to the insurer if commitment C9 is created for repairing the vehicle. In C13, the insurer commits to the repairer for payment if the vehicle is repaired. The commitment C14 means that the insurer commits to the assessor for payment if the assessor brings about the agreement to repair.

There is no commitment associated with some of the task dependencies. For example, the assessor depends upon the insurer for sending a claim for assessment. In this case, the insurer does not commit to sending the claim to the assessor.

4 Verifying Agent Interactions

Since agents are autonomous (based on the fact that they represent autonomous business organizations), they may not conform to a given model. So it is important to verify agent interactions with respect to a model. A business model captures commitments between agents. The commitments provide a basis for verifying agent interactions for conformance with the specified business model.

We consider a UML sequence diagram as a low-level model for agent interactions as they are realized. The roles appear as objects in this diagram and they exchange messages. Roles may exchange multiple messages for executing one task. For example, consider the task of reporting an accident. The policy holder sends a message to the insurer for this task. If the information in that message is incomplete, the insurer may send a message to the policy holder requesting additional information. This would repeat until the insurer receives all information, at which point the task of reporting accident would complete.

An agent conforms to a business model if it satisfies each commitment of which it is the debtor and whose antecedent holds. To verify conformance, we iterate over active commitments from the business model. For each commitment, we evaluate its antecedent and consequent using the tasks and the domain facts asserted in the interaction model. Each commitment whose consequent evaluates to true is satisfied. Each commitment whose antecedent evaluates to true, but consequent to false, is a detached commitment that is violated. Hence, the debtor of a violated commitment is the agent that fails conformance with respect to the business model.

The agent interactions can be verified either at design time or at run time. At design time, a low-level interaction model design can be verified against a business model. There can be many interaction models that satisfy a given business model. At run time, the emergent agent behavior, captured in the form of a low-level interaction model, can be verified against a business model. When such verification is performed, some interactions may still be pending, and therefore, some commitments may be eventually satisfied. Therefore, to detect violations we must model the various tasks as being time bounded, that is, as including timeouts. A commitment whose antecedent evaluates to true but consequent to false (taking timeouts into consideration) is violated. Figure 16 shows a series of conforming and nonconforming interactions for AGFIL scenario. A message labeled *Ti* corresponds to the task *i* in Table 2.

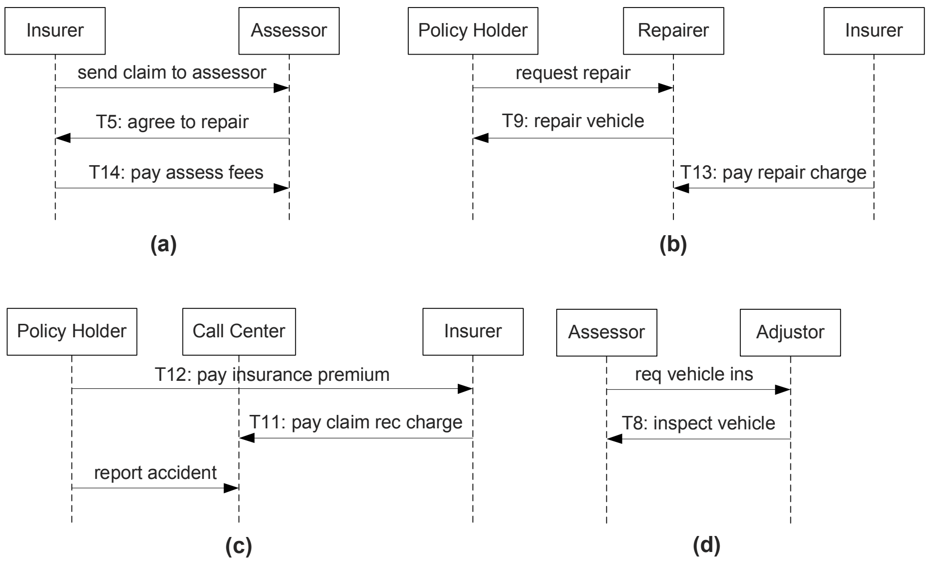


Fig. 16. Conforming and nonconforming agent interactions

- Fig. 16(a) shows an interaction between the insurer and the assessor. The insurer sends a claim for assessment to the assessor. The assessor negotiates the repair charge with the repairer, and brings about an agreement with the repairer for vehicle repair. This satisfies commitment C5, and detaches commitment C14. The insurer pays the assessment fees to the assessor, and satisfies C14. Since, the insurer and the assessor satisfy their commitments, they conform to the business model.
- Fig. 16(b) shows another example of conforming interaction between a policy holder, a repairer, and the insurer. The policy holder provides claim information, and requests the repairer for vehicle repair. The repairer finds the claim to be valid, and repairs the vehicle. The repairer satisfies commitment C9 and detaches commitment C13. By paying the repair charge, the insurer satisfies commitment C13.
- Fig. 16(c) shows an example of nonconforming interaction between a policy holder, the call center, and the insurer. The policy holder obtains insurance by paying the requisite premium to the insurer. The insurer pays the call center for providing claim reception service. Then the policy holder reports an accident and detaches commitments C1, C2, and C4, assuming the claim to be valid. But the call center does not gather the claim information, assign a garage, or send the claim to the insurer. Therefore, the call center violates commitments C1, C2, and C4, and does not conform to the model.
- Fig. 16(d) shows another example of a nonconforming interaction. The assessor requests a vehicle inspection from the adjustor. The adjustor inspects the vehicle, and detaches commitment C10. However, the assessor violates commitment C10 by not paying the inspection fees to the adjustor, and therefore fails to conform with the model.

5 Discussion

This paper proposes an agent-oriented business metamodel based on Tropos. The model uses the mental and social concepts of Tropos. It also uses the goal, plan, and dependency modeling techniques from Tropos. The goal and plan modeling are based on the means-end and AND-OR analyses.

Our approach offers two major benefits. One, the high-level metamodel captures the business relationships directly. Thus it shows how the business model may be modified in the face of changing business needs. For example, the insurer may decide to outsource claim handling (as in the above case) or may decide to insource it. A business analyst can readily determine if the resulting business model is sound. Likewise, each participant can evaluate a potential change to the business model in terms of whether it would affect the commitments of which it is the debtor or the creditor. Two, the high-level metamodel yields a natural basis for reasoning about correctness. We can use the business relationships as a basis for determining whether a particular enactment is conforming and whether a particular way to generate an enactment is sound.

Tropos is a general purpose agent-oriented software engineering methodology. It can be applied to a wide range of software applications, and it covers all phases of software development. In contrast, our proposed methodology is tailored specifically for business modeling.

A key difference between our model and Tropos is the concept of commitment. In Tropos, a dependency means that a depender actor depends on a dependee actor for executing a plan or achieving a goal. The concept of dependency does not model what is required of the depender, and the dependee unconditionally adopts the dependency. The debtor, creditor, and consequent of a commitment are similar to the Tropos dependee, depender, and dependum, respectively. However, unlike a dependency, a commitment includes an antecedent that brings it into full force. This enables modeling reciprocal relationships between economic entities, which is lacking in the concept of dependency.

Andersson *et al.* [1] present what they call a "reference" ontology for business models based on concepts from three approaches, namely, REA, BMO, and e^3-value. Table 3 compares their ontology concepts to the concepts from our model. Their concepts of actor and actor type are similar to our agent and role, respectively. A domain ontology captures resource, resource type, feature, and right in our approach. Our task abstraction is similar to the concept of event without additional classification into types. Andersson *et al.*'s notion of commitment is close to our concept of commitment. Their concepts of exchange, transaction, contract, agreement, and reciprocity are reflected as two or more commitments in our approach. Unlike our metamodel, Andersson *et al.*do not model actor goals.

Gordijn and Wieringa [5] propose e^3-value business model. Unlike our metamodel, e^3-value is a semiformal model based on economic concepts, and is mainly intended for profitability analysis. A value interface in e^3-value aggregates related in and out value ports of an actor to represent economic reciprocity. This concept is close to our concept of commitment, but it lacks similar semantics and flexibility. For example, unlike a value interface, a commitment can be delegated.

Table 3. Proposed business metamodel related to Andersson *et al.*'s ontology

Reference ontology concept	**Business metamodel concept**
Actor	Agent
Actor type	Role
A pair of transfers	A pair of commitments
Resource, resource type, feature, right	Defined in domain ontology
Event	Task
Commitment	Commitment
Exchange	A pair of commitments
Transaction	Set of commitments
Contract	Set of commitments
Agreement	Commitment
Reciprocity	A pair of commitments
Claim	Detached commitment

Due to this, an e^3-value model may capture value exchange among two actors, but during execution, the exchange and interaction may take place between two different actors, and without a clear notion of delegation, it is not obvious how the latter is selected.

Opera is a framework for modeling multi-agent societies [7], though from the perspective of a single designer or economic entity. In contrast, we model interactions among multiple entities. Opera's concepts of landmark and contract are close to our concepts of task and commitment, respectively. However, Opera uses traditional obligations, which lack the flexibility of commitments. Unlike obligations, a commitment can be manipulated as Sec. 2 describes.

Amoeba [4] is a process modeling methodology based on commitment protocols. This methodology creates a process model in terms of fine-grained messages and commitments. In contrast, our model is at a higher level of abstraction and includes business goals and tasks in addition to commitments.

References

1. Andersson, B., Bergholtz, M., Edirisuriya, A., Ilayperuma, T., Johannesson, P., Gordijn, J., Grégoire, B., Schmitt, M., Dubois, E., Abels, S., Hahn, A., Wangler, B., Weigand, H.: Towards a reference ontology for business models. In: Embley, D.W., Olivé, A., Ram, S. (eds.) ER 2006. LNCS, vol. 4215, pp. 482–496. Springer, Heidelberg (2006)
2. Bresciani, P., Perini, A., Giorgini, P., Giunchiglia, F., Mylopoulos, J.: Tropos: An agent-oriented software development methodology. Autonomous Agents and Multi-Agent Systems 8(3), 203–236 (2004)
3. Browne, S., Kellett, M.: Insurance (motor damage claims) scenario. Document Identifier D1.a, CrossFlow Consortium (1999)

4. Desai, N., Chopra, A.K., Singh, M.P.: Amoeba: A methodology for modeling and evolution of cross-organizational business processes. ACM Transactions on Software Engineering and Methodology, TOSEM (to appear, 2009)
5. Gordijn, J., Wieringa, R.: A value-oriented approach to E-business process design. In: Eder, J., Missikoff, M. (eds.) CAiSE 2003. LNCS, vol. 2681, pp. 390–403. Springer, Heidelberg (2003)
6. Singh, M.P.: An ontology for commitments in multiagent systems: Toward a unification of normative concepts. Artificial Intelligence and Law 7, 97–113 (1999)
7. Weigand, H., Dignum, V., Meyer, J.-J.C., Dignum, F.: Specification by refinement and agreement: Designing agent interaction using landmarks and contracts. In: Petta, P., Tolksdorf, R., Zambonelli, F. (eds.) ESAW 2002. LNCS, vol. 2577, pp. 257–269. Springer, Heidelberg (2003)

"Reducing" CLASSIC to Practice: Knowledge Representation Theory Meets Reality*

Ronald J. Brachman[1], Alex Borgida[2,**],
Deborah L. McGuinness[3], and Peter F. Patel-Schneider[4]

[1] Yahoo! Research
[2] Rutgers University
[3] Rensselaer Polytechnic Institute
[4] Bell Labs Research

Abstract. Most recent key developments in research on knowledge representation (KR) have been of the more theoretical sort, involving worst-case complexity results, solutions to technical challenge problems, etc. While some of this work has influenced practice in Artificial Intelligence, it is rarely—if ever—made clear what is compromised when the transition is made from relatively abstract theory to the real world. CLASSIC is a description logic with an ancestry of extensive theoretical work (tracing back over twenty years to KL-ONE), and several novel contributions to KR theory. Basic research on CLASSIC paved the way for an implementation that has been used significantly in practice, including by users not versed in KR theory. In moving from a pure logic to a practical tool, many compromises and changes of perspective were necessary. We report on this transition and articulate some of the profound influences practice can have on relatively idealistic theoretical work. We have found that CLASSIC has been quite useful in practice, yet still strongly retains most of its original spirit, but much of our thinking and many details had to change along the way.

FORWARD - February 2009

The practical instantiation of theories is an important, but often neglected, component of knowledge representation. The work on CLASSIC, described in this paper, was undertaken at AT&T Bell Labs around 1990, and fits into this component, taking the formal theories of description logic and crafting a useful system for representing information.

The state of the art in description logics has changed considerably since CLASSIC was first envisioned in the late 1980s. Advances in algorithms and computers have made for effective reasoners for theoretically intractable description logics,

* A slightly different version published in *Artificial Intelligence*, 114, October 1999, pages 203–237.

** Supported in part by the AT&T Foundation and NSF IRI 9119310 & 9619979.

leading up to modern reasoners for description logics like SHOIQ, where reasoning is NExpTime complete.

These advances in reasoning have led to the adaptation of description logics to the Semantic Web. The changes needed were theoretically quite small, amounting mostly to using IRIs as names and adding the datatypes of XML Schema. This lead, through a number of intermediate steps, to the W3C Web Ontology Language OWL. However, quite a bit of practical adaptation was needed as well, including allowing non-logical decorations, adding a method of combining documents on the web into one knowledge base, building user interface systems like Protégé (http://www.co-ode.org/downloads/protege-x/), and providing APIs for popular programming languages. There are a now multiple reasoning systems for OWL, including Pellet (http://clarkparsia.com/pellet/), Racer (http://www.racer-systems.com/), and FaCT++ (http://code.google.com/p/factplusplus/), which are based on this combination of theoretical underpinnings and attention to practical details, now prevalent in the description logic community.

1 Introduction

In recent years, the research area of knowledge representation (KR) has come to emphasize and encourage what are generally considered more "theoretical" results, such as novel logics, formal complexity analyses, and solutions to highly technical challenge problems. In many respects, this is reasonable, since such results tend to be clean, presentable in small packages (i.e., conference-length papers), easily built upon by others, and directly evaluable because of their formal presentation. Even the recent reappearance of papers on algorithms and empirical analyses of reasoning systems has emphasized the formal aspects of such work. But lurking in the virtual hegemony of theory and formalism in published work is the tacit belief that there is nothing interesting enough in the reduction of KR theory to practice to warrant publication or discussion. In other words, it seems generally assumed that theoretical contributions can be reduced to practice in a straightforward way, and that once the initial theoretical work is wrapped up, all of the novel and important work is done.

Experience in building serious KR systems challenges these assumptions. There is a long and difficult road to travel from the pristine clarity of a new logic to a system that really works and is usable by those other than its inventors. Events along this road can fundamentally alter the shape of a knowledge representation system, and can inspire substantial amounts of new theoretical work.

Our goal here is to attempt to show some of the key influences that KR practice can exert on KR theory. In doing so, we hope to reveal some important contributions to KR research to be made by those most concerned with the reduction to practice. In order to do this, we will lean on our experience with CLASSIC, a description logic-based KR system. CLASSIC has seen both ends of the theory-to-practice spectrum in quite concrete ways: on the one hand, it was developed after many years of KL-ONE-inspired research on description systems, and was initially presented as a clean and simple language and system [5] with a formal semantics for subsumption, etc.; on the other, it has also been re-engineered from an initial LISP implementation to C and then to C++, has been widely distributed, and has been used in several fielded industrial products in several companies on an everyday basis (making CLASSIC one of a few KR systems to be moved all the way into successful commercial practice). Our substantial system development effort made it clear to us that moving a KR idea into real practice is not just a small matter of programming, and that significant research is still necessary even after the basic theory is in place.

Our plan in this paper is first to give an introduction to the goals and design of our system, and then to summarize the "theoretical" CLASSIC as originally proposed and published. We will then explain our transition from research design to production use. Rather than give a potentially unrevealing historical account, we will subsequently attempt to abstract out five general factors that caused important changes in our thinking or in the system design, giving concrete examples of each in the evolution of CLASSIC:

1. general considerations in creating and supporting a running system;
2. implementation issues, ranging from efficiency tradeoffs to the sheer complexity of building a feature;
3. general issues of real use by real people, such as learnability of the language and error-handling;
4. needs of particular applications; and
5. the unearthing of incompleteness and mistakes through programming, use, and community discussion.

While some of these concerns may seem rather prosaic, the point is that they have a hitherto unacknowledged—and substantial—influence on the ultimate shape, and certainly on the ultimate true value of any knowledge representation proposal. There is nothing particularly atypical about the research history of CLASSIC; as a result, the lessons to be learned here should apply quite broadly in KR research, if not in AI research in general.

2 The Original CLASSIC

While potential applications were generally kept in mind, CLASSIC was originally designed in a typical research fashion—on paper, with attention paid to formal, logical properties, and without much regard to implementation or application details. This section outlines briefly the original design, before we attempted to build a practical tool based on it. The interested reader can also see [5], which is a traditional research article on the original version of the language, and [12] for more details.

2.1 Goals

CLASSIC was designed to clarify and overcome some limitations in a series of knowledge representation languages that stemmed originally from work on KL-ONE [9,13]. Previous work on NIKL [32], KRYPTON [10], and KANDOR [36] had shown that languages emphasizing structured descriptions and their relationships were of both theoretical and practical interest. Work on KL-ONE and its successors[1] grew to be quite popular in the US and Europe in the 1980's, largely because of the semantic cleanliness of these languages, the appeal of object-centered ("frame") representations, and their provision for some key forms of inference not available in other formalisms (e.g., *classification*—see below). Numerous publications addressed formal and theoretical issues in "KL-ONE-like" languages, including formal semantics and computational complexity of variant languages (see, or example, [19]). However, the key prior implemented systems all had some fundamental flaws,[2] and the CLASSIC effort, initiated in the mid '80's at AT&T, was in large part launched to design a formalism that was free of these defects.

Another central goal of CLASSIC was to produce a compact logic and ultimately, a small, manageable, and efficient system. Small systems have important advantages in a practical setting, such as portability, maintainability, and comprehensibility. Our intention was eventually to put KR technology in the hands of regular technical (non-AI) employees within AT&T, to allow them to build their own domain models and maintain them. Success on this account seemed to very strongly depend on how simply and effortlessly new technology could integrate into an existing environment and on how easy it would be to learn how to use it, not to mention on reasonable and predictable performance. Our plan was to take computational predictability (both of the inferences performed and the resources used) seriously. All in all, because of our desire to connect with extremely busy developers working on real problems, simplicity and performance were paramount in our thinking from the very beginning.

As has been often discussed in the literature [11,21], expressiveness in a KR language trades off against computational complexity, and our original hope was to produce a complete inference system that was worst-case tractable. This contrasts the CLASSIC work even with other contemporaneous efforts in the same family, e.g., LOOM [24] and BACK [39], and with much work elsewhere in KR. Like CLASSIC, both of these systems ended up in the hands of users at sites other than where they were developed (in fact, LOOM has a user base of several hundreds, mostly in the AI research community), and had to go through the additional efforts, described in this paper, at supporting a user interface, escape mechanisms from expressive limitations, and rules. However, both LOOM and BACK opted for more expressive logics and incomplete implementations, which left them with the problem of characterizing the inferences (not) being performed. In CLASSIC, although we eventually incorporated language features that were on paper intractable (e.g., role hierarchies), the implementation of concept reasoning was kept complete, except for reasoning with individuals in concepts, and in that case both a precise formal and procedural characterization were given [7].

[1] These systems came to be called "description logics"—see below.

[2] E.g., NIKL had no mechanism for asserting specific, concrete facts; KRYPTON was so general as to be fairly unusable; and KANDOR handled individual objects in a domain in a very incomplete way.

Many more recent description logic-based KR systems have explicitly taken a different view of this expressiveness/tractability tradeoff. They chose to implement complete reasoners for expressive languages and thus, of necessity, have intractable inference algorithms. Initially these systems had poor computational properties [2,15], but recent advances in description logic inference algoritms, initiated by Ian Horrocks, have led to a new generation of systems with surprisingly effective performance [22,38].

CLASSIC continues to be an important data point, and a useful experiment on exactly how one can design a small but useful language, but as we see below, the original theoretical goal of worst-case polynomial complexity could not be preserved without detriment to real users.[3]

Finally, CLASSIC was designed to fill a small number of specific application needs. We had had experience with a form of deductive information retrieval, for example in the context of information about a large software system [18], and needed a better tool to support this work. We also had envisioned CLASSIC as a deductive, object-oriented database system (see [5]; some success on this front was eventually reported in [42] and [14]). It was *not* our intention to provide some generic "general knowledge representation system," applicable to any and all problems. CLASSIC would probably not be useful, for example, for arbitrary semantic representation in a natural language system, nor was it intended as a "shell" for building an entire expert system. But the potential gains from keeping the system small and simple justified the inability to meet arbitrary (and too often, ill-defined) AI needs—and it was most important to have it work well on our target applications.

2.2 The Description Language

CLASSIC is based on a *description logic* (DL), a formal logic whose principal objects are structured terms used to describe individual objects in a domain. Descriptions are constructed by composing *description-forming constructors* from a small repertoire and specifying arguments to these constructors. For example, using the constructor **ALL**, which is a way to restrict the values of a property to members of a single class, we might construct the description, "(something) all of whose children are female":

```
(ALL child FEMALE).
```

Descriptions can then be asserted to hold of individuals in the domain, associated with names in a knowledge base, or used in simple rules. Because of the formal, compositional structure of descriptions (i.e., they are like the complex types in programming languages), certain inferences follow both generically—one

[3] However, we state emphatically that this did not mean that the only alternative was to resort to an arbitrarily expressive language to satisfy our users. As we discuss below, the extensions made to meet real needs were generally simple, and CLASSIC is still clearly a small system, yet one of the most widely used description logic-based systems.

description can be proven to imply another—and with respect to individuals—a description can imply certain non-obvious properties of an individual.

In CLASSIC, as in most description logic work, we call our descriptions *concepts*, and individual objects in the domain are modeled by *individuals.* To build concepts, we generally use descriptions of properties and parts, which we call *roles* (e.g., `child`, above). CLASSIC also allows the association of simple *rules* with named concepts; these were considered to be like forward-chaining procedural rules (also known as "triggers" in active databases). For example,

```
AT&T-EMPLOYEE ⇝
(AND (AT-LEAST 1 HRID-number)
     (ALL HRID-number 7-DIGIT-INTEGER))
```

would mean "If an AT&T employee is recognized then assert about it that it has at least one HR ID number, all of which are 7-digit integers."

CLASSIC's description-forming constructors were based on the key constructs seen in frame representations over the years. These constructors have cleaned up ambiguities in prior frame systems, and are embedded in a fully compositional, uniform description language. The constructors in the original design ranged from conjunction of descriptions (**AND**); to role "value restrictions" (**ALL**); number restrictions on roles (**AT-LEAST**, **AT-MOST**); a set-forming constructor (**ONE-OF**); and constructors for forming "primitive" concepts (**PRIMITIVE**, **DISJOINT-PRIMITIVE**), which have necessary but not sufficient conditions. CLASSIC also has a constructor **SAME-AS**, specifying objects for which the values of two sequences of roles have the same value; and a role-filling constructor (**FILLS**) and a constructor for "closing" roles (**CLOSE**),[4] although these were originally only applicable to individuals.

2.3 Operations on CLASSIC Knowledge Bases

The description-forming constructors are used to create descriptions that are, in turn, used to define named concepts, create rules, and describe individual objects in the domain. In order to create new concepts and assign descriptions to individuals, etc., the user interacts with the CLASSIC system by means of a set of *knowledge base-manipulating operations.* Generally speaking, operations on a CLASSIC knowledge base (KB) include additions to the KB and queries.

Additive (monotonic) updates to the KB include the definition of new concepts or roles, specification of rules, and assertion of properties to hold of particular individuals. By "definition" here, we mean the association of a name (e.g., `SATISFIED-GRANDPARENT`) with a CLASSIC description (e.g., `(`**AND** `PERSON (`**ALL** `grandchild MARRIED))`, intended to denote "a person all of whose grandchildren are married"). One of the contributions of CLASSIC, also developed

[4] Role fillers for CLASSIC's individuals are treated under a "non-closed-world" assumption, in that unless the KB is told that it knows all the fillers of a certain role, it assumes that more can be added (unless the number restrictions forbid it, in which case role closure is implied).

contemporaneously in BACK [43], is the ability to specify incomplete information on individuals; for example, it is possible to assert that `Leland` is an instance of

```
(AND (AT-LEAST 1 child)
     (ALL works-for
          (ONE-OF Great-Northern-Hotel Double-R-Diner)))
```

indicating that “Leland has at least one child and works for either the Great Northern Hotel or the Double-R Diner.” Thus, individual objects are not required to be in the restricted form of simple tuples or complete records.

CLASSIC can also answer numerous questions from a KB, including whether one concept subsumes another, whether an individual is an instance of a concept, and whether two concepts are disjoint, and it can respond to various types of retrieval queries (e.g., fetch the properties of an individual, fetch all the instances of a concept).

These general operations on CLASSIC knowledge bases were characteristic of the system throughout its evolution, although specific operations changed in interesting ways and new ones (especially dealing with retraction of information) were added, as discussed later.

2.4 Inferences

A key advantage of CLASSIC over many other systems was the variety and types of inferences it provided. Some of the standard frame inferences, like (strict) inheritance, are part of CLASSIC’s definition, but so are several others that make description-logic-based systems unique among object-centered KR systems. Among CLASSIC’s inferences are

– *“completion” inferences*: logical consequences of assertions about individuals and descriptions of concepts are computed; there are a number of these inferences CLASSIC can make, including *inheritance* (if A is an instance of B and B is a subclass of C, then A “inherits” all the properties of C), *combination* of restrictions on concepts and individuals, and *propagation* of consequences from one individual to another (if A fills a role r on B, and B is an instance of something which is known to restrict all of its fillers for the r role to be instances of D, then A is an instance of D);
– *contradiction detection*: inherited and propagated information is used to detect contradictions in descriptions of individuals, as well as incoherent concepts.
– *classification and subsumption inferences*: *concept classification*, in which all concepts more general than a concept and all concepts more specific than a concept are found; *individual classification*, in which all concepts that an individual satisfies are determined; and *subsumption*, i.e., whether or not one concept is more general than another.
– *rule application*: when an individual is determined to satisfy the antecedent of a rule, it is asserted to satisfy the consequent as well.

2.5 Other Theoretical Aspects of CLASSIC

Since the original design of CLASSIC proceeded mainly from a theoretical standpoint, other formal aspects of the logic were explored and developed. CLASSIC's concept constructors had a traditional formal semantics similar to the semantics of related languages. Since reasoning about individuals was more procedural, and did not have the same formal tradition, it did not initially receive the same precise formal treatment. In hindsight, it would have been of considerable help to have found a formal semantics for this aspect too, although this ended up requiring entirely new machinery (as discovered by Donini, *et al.* [20]) and may well have cost us too much in terms of time and lost opportunities (another key factor in the production of a successful system).

Given our desire for compactness and performance, we were also concerned with the computational complexity of the inferences to be provided by CLASSIC. Based on the experience with implementing previous description logic-based systems, and the work of Aït-Kaci [1] on reasoning with attribute-chain identities, we believed that we had a good chance to develop an efficient, polynomial-time algorithm for the subsumption inference. However, at this stage we did not have a formal proof of the tractability of the logic, nor of the completeness of our implementation. To some extent we were going on faith that the language was simple enough, and that we were avoiding the known intractability traps, such as those pointed out by Nebel [34]. As we shall see, this faith was in part misplaced, from the strictly formal point of view, although in practice we were on the right track.

2.6 Anticipating Implementation

The original CLASSIC proposal to some extent anticipated building a practical tool. Two concessions were made directly in the language. First, inspired by the TAXIS database programming language [33], we allowed the formation of concepts that could compute their membership using *test functions* to be written in the host programming language. **TEST** concepts would act as primitive concepts, in that their necessary and sufficient conditions would not be visible to CLASSIC for inference. But they would allow the user to partly make up for CLASSIC's limited expressive power, at least in dealing with individuals.

Second, we specified a class of individuals in the language called *"host" individuals*, which would allow direct incorporation of things like strings and numbers from the host programming language. Many previous KR languages had failed to make a clean distinction between values, such as numbers and strings, borrowed directly from the implementation, and objects totally within the representation system. CLASSIC cleared this up even in the initial formal design; for example, a host individual could not have roles, since it is immutable and all of its properties are implied by its identity. Also, the (test) definitions of important host concepts (like `NUMBER`, `STRING`, etc.) could be derived automatically from the host environment.

2.7 The Result

Prior to the completion and release of an implementation and its use in applications, CLASSIC thus had the description language grammar illustrated in Figure 1—this is roughly the version of CLASSIC published in 1989 [5], and was the initial basis for discussions with our development colleagues (see below). Using descriptions formed from this grammar, a user could create a knowledge base by defining new named concepts and roles, asserting that certain restricted types of descriptions applied to individuals, and adding rules, as defined in Figure 2.

Numerous straightforward types of KB queries were also available. These included retrieval of information told to the system, retrieval of information derived by the system, and queries about the derived taxonomy of concepts and individuals.

Even at this point, the CLASSIC logic made a number of new contributions to KR, including the following:

– full integration of host-language values,
– support for equalities on attribute chains (**SAME-AS**),
– the ability to assert partial information about individuals,
– the addition of a simple form of forward-chaining implication to a KL-ONE-like language,
– the **ONE-OF** construct for constructing explicit sets of individuals,
– a broad cadre of inferences including full propagation of information implied by assertions, and
– a comprehensive and clean retraction mechanism for assertions,

```
<concept-expression> ::=
        THING | CLASSIC-THING | HOST-THING |                      % built-in names
        <concept-name> |                                 % names defined in the KB
        (AND <concept-expression>+) |                               % conjunction
        (ALL <role-name><concept-expression>) |      % universal value restriction
        (AT-LEAST <positive-integer><role-name>) |          % minimum cardinality
        (AT-MOST <non-negative-integer><role-name>) |       % maximum cardinality
        (SAME-AS (<role-name>+)(<role-name>+)) |            % role-filler equality
        (TEST <function> [<realm>]) |                            % procedural test
        (ONE-OF <individual-name>+) |                        % set of individuals
        (PRIMITIVE <concept-expression> <index>) |            % primitive concept
        (DISJOINT-PRIMITIVE <concept-expression> <group-index> <index>)
<realm> ::= HOST | CLASSIC
<individual-expression> ::=
        <concept-expression> |
        (FILLS <role-name> <individual-name>) |                     % role-filling
        (CLOSE <role-name>) |                                       % role closure
        (AND <individual-expression>+)                               % conjunction
```

Fig. 1. The Original CLASSIC Expression Grammar (comments in italics)

```
<knowledge-base> ::= <statement>+
<statement> ::=
        (DEFINE-CONCEPT <name> <concept-expression>)          % new concept
        (DEFINE-ROLE <name>)                                   % new role
        (CREATE-IND <name> <individual-expression>)         % new individual
        (IND-ADD <name> <individual-expression>)           % add information
        (ADD-RULE <concept-name> <concept-expression>)            % add a rule
```

Fig. 2. The Original CLASSIC Knowledge Base Grammar

all in the context of a simple, learnable clean frame system.[5] In this respect, the work was worth publishing; but in retrospect it was naive of us to think that we could “just” build it and use it.

3 The Transition to Practice

Given the simplicity of the original design of CLASSIC, we held the traditional opinion that there was essentially no research left in implementing the system and having users use it in applications. In late 1988, we concluded a typical AI programming effort, building a CLASSIC prototype in COMMON LISP. As this version was nearing completion, we began to confer with colleagues in an AT&T development organization about the potential distribution of CLASSIC within the company. Despite the availability of a number of AI tools, an internal implementation of CLASSIC held many advantages: we could maintain it and extend it ourselves, in particular, tuning it to our users; we could assure that it integrated with existing, non-AI environments—our many legacy systems; and we could assure that the system had a well-understood, formal foundation (in contrast to virtually all AI tools commercially available at the time). Thus we initiated a collaborative effort to create a truly practical version of CLASSIC, written in C. Our intention was to develop the C version, maintain it, create a training course, and eventually find ways to make it usable even by novices. Meanwhile, as the C effort progressed, we began to experiment with our first applications using the LISP prototype.

The issues and insights reported in the next several sections arose over an extended period of time, in which we collaborated on the design of the C version of CLASSIC, and developed several substantial and different types of applications, and, later, produced a third version of CLASSIC (in C++). Two of the applications were more or less along the lines we expected. But the others were not originally foreseen. The first two applications—a software information system using classification to do retrieval [18], and what could be considered a “semantic data model” front end to a relational database of program cross-reference

[5] Some, though not all, of these features were independently introduced in the other two description logic-based systems being developed at about the same time—LOOM [24] and BACK [39].

information [42]—were planned from the beginning. Other types of applications were unanticipated. One family of applications (FindUR [26]) used CLASSIC to manage background ontologies, which were used to do query expansion in World-Wide Web searches. The description logic system organized descriptions and detected problems in collaboratively generated ontologies maintained by people not trained in knowledge representation. Another application proved to be the basis of a commercially deployed NCR product. It was the front end for data analysis in a data mining setting and was marketed as the "Management Discovery Tool" (MDT). Our family of configuration applications [44,31,30,28] was the most successful of the applications. Fifteen different configurators were built using CLASSIC, and these have gone on to have processed more than $6 billion worth of orders for telecommunications equipment in AT&T and Lucent Technologies.

In addressing the needs of these applications, we developed interim versions of the LISP CLASSIC system and received feedback from users along the way. Thus the "transition research" reported here was stimulated both by the need to construct a usable system in general and by the demands of real users in real applications.

4 Feedback from Practice

The process of implementing, re-implementing in C, and having application builders use CLASSIC and provide us with feedback, resulted in significant changes. Some of these arose because of simple facts of life of providing real software to real people, some arose because it is impossible to anticipate many key needs before people start to use a system, and some arose because it is hard to truly complete the formal analysis before you begin building a system.[6]

There are many significant lessons to be learned from the attempt to move an abstract KR service into the real world, some of which are sociological and very general, and some of which are very low-level and relate only to implementation details. We restrict ourselves here mainly to technical considerations that influenced the ultimate shape of the CLASSIC language and KB operations, and critical differences between the original proposal and the final form. We look at a small number of examples in each case. Even looking at the feedback from practice to theory involving only basic language and KB operations, there are numerous factors that end up shaping the final version of a logic. Here we cover issues relating to software system development, *per se*; implementation considerations; needs of users; the needs of specific applications; and the demand from practice that all details be worked out.

[6] This is by no means to say that such prior formal analyses are not also critical to the success of a project. Here, our intention is simply to focus on the relatively unheralded role that implementation and use play in the success and ultimate shape of KR systems.

4.1 Creating and Supporting a System

Even if the setting is not a commercial one, the intent to create and release a system carries certain obligations. In the case of CLASSIC, our development collaborators made one thing very clear: do not release software of which you are unsure. In particular, it is important not to include features in an initial release that you might choose later to remove from the language. Once someone starts to use your system, it is almost unpardonable to release a future generation in which features that used to be supported no longer are (at least with respect to the central representation language constructs).

Even in a research setting, while we are used to playing with features of a language to see what works best, once software is created, it is awkward at best to nonmonotonically change the supported language. In CLASSIC, this meant that we needed to carefully consider every constructor, to make sure that we were confident in its meaning and utility. In particular, we were forced by our commitment to efficient implementation to exclude the original, general **SAME-AS** constructor from the initial system specification because its implementation was appearing increasingly more complex (see Section 4.2).

This constraint also forced us to abandon a constructor we were considering that would allow the expression of concepts like "at least one female child" (this type of constuct has come to be known as a "qualified number restriction"). In the general case, such a constructor rendered inference intractable [34] and we wanted to avoid known intractability. Had our users demanded these constructors, we might have tried for a reasonable partial implementation. Unfortunately, we did not have a handle on an incomplete algorithm that we could guarantee would only get more complete with subsequent releases, and which had a simple and understandable description of what it computed. The latter is particularly important, since otherwise users might expect certain conclusions to be drawn when the algorithm in fact would miss them. Such mismatched expectations could be disastrous for the acceptability of the product. We were strongly told that it was better to eliminate a construct than to have a confusing and unintuitive partial implementation of it. Upward compatibility dictated that even with incomplete algorithms, subsequent versions of the system would still make at least all inferences made in earlier versions. Thus, we were better off from this perspective in keeping the language simple, and left this construct out of the initial released version.

There were other influences on the evolution of CLASSIC because of this prosaic, but important type of constraint. For a while, we had contemplated a role inverse construct (see Section 4.2). For example, as was common in many older semantic network schemes, we wanted to say that `parent` was the inverse of `child`. While we had not designed such a construct into the original specification, it appeared to be potentially very useful in our applications. We worked out several solutions to the problem, including a fairly obvious and general one that allowed inverses to be used for any role at any time. However, the cost appeared to be very high, and it was not even clear that we could implement the most general approach. As a result, we were forced to abandon any attempt to implement role inverses in the first released version of CLASSIC. We were not totally confident that we could stick with any initial attempt through all subsequent releases, and we did not want to start with an awkward compromise that might be abandoned later.

Another consideration is harder to be technical about or to quantify, but was equally important to us. As the system began to be used, it became clear that we needed to be as sure as possible that our constructors were the best way to carve up the terminological space. In description logics, there are many variant ways to achieve the same effect, some of which are more elegant and comprehensible than others. Since we intended to put our software in the hands of non-experts, getting the abstraction right was paramount. Otherwise, either the system would simply not be used, or a second release with better constructors would fail to be upward compatible. In CLASSIC's case, we put a great deal of effort into an assessment of which constructors had worked well and which had not in previous systems.

Finally, as mentioned, CLASSIC early on became a coordinated effort between research and development organizations. Once we had the development (C language) version of CLASSIC released and a set of commercial applications in place, we thought that we had reached a steady state. The research team would experiment with adding new features first to the LISP version of CLASSIC, and then the development team would port those features to the C version. Both teams were involved in both activities, so that while the C version of CLASSIC would lag behind the LISP version, it would never be too far behind.

Unfortunately, this anticipated mode of operation turned out to have several problems. First, for pragmatic reasons in getting off the ground, the C version never did have all the features of even the initial LISP version. Second, once the C version was suitable for the commercial applications, there was no short-term development reason for adding features to it. Additions to the C version would be supported by development resources only in response to needs from current or proposed applications. Third, the research team had neither the resources nor, indeed, the expertise to change the C version.

These mundane human resource constraints meant that it was very unlikely that the C version of CLASSIC would ever approach the capabilities of the LISP version. Once we realized this, we decided that the only solution would be to create a combined version of CLASSIC in a language acceptable for both research and development. This combined version, which we called NeoClassic, was written in C++, the only language acceptable to development that was reasonable for research purposes. NeoClassic was designed to immediately supplant the C version of CLASSIC and was supposed to quickly acquire the features of the LISP version, modified as appropriate. Implementation of NeoClassic proceeded to the point that it was usable and as featureful as the C version, but ultimately corporate changes and personnel issues caused the work to be scaled back.

4.2 Implementation Considerations

There were at least two aspects of the implementation effort itself that ultimately ended up influencing the language and operation design. One was the sheer difficulty of implementing certain inferences, and the other was the normal kind of tradeoff one makes between time and space.

Complex Operations. We began our implementation by attempting to construct a complex subsumption algorithm that included **SAME-AS** relations between roles. For example, we wanted to be able to detect that a concept that included

```
(AND (SAME-AS (boss) (tennis-partner))
     (SAME-AS (tennis-partner) (advisor)))
```

was subsumed by (i.e., was more specific than) a concept that included

```
(SAME-AS (boss) (advisor));
```

in other words, that someone whose boss is her tennis partner, and whose tennis partner is her advisor, must of necessity be someone whose boss is her advisor.

Because **SAME-AS** could take arbitrary role *paths*, roles could possibly be unfilled for certain individuals, and roles could have more than one filler; this made for some very complex computations in support of subsumption. Cases had to be split, for example, into those where roles had some fillers and those where they had none, and less-than-obvious multiplications and divisions had to be made in the presence of number restrictions. More than once, as soon as we thought we had all cases covered, we would discover a new, more subtle one that was not covered. When we finally came to a point where we needed a result from group theory and the use of factorial to get some cardinalities right, we decided to abandon the implementation.

As it turns out, our attempts at a straightforward and efficient solution to what appeared to us to be a tractable problem were thwarted for a deep reason: full equality for roles is undecidable. This result was later proved by Schmidt-Schauss [41] and Patel-Schneider [37]. Thus, our implementation enterprise could *never* have fully succeeded, although we did not know it at the time. The end result of all of this was an important change to the CLASSIC language (and therefore needed to be reflected in the semantics): we moved to distinguish between *attributes*, which could have exactly one filler, and other roles, which could have more than one filler. **SAME-AS** could then be implemented efficiently for attributes (using ideas from [1]), appropriately restricted to reflect the dichotomy. In the end, the distinction between attributes and multiply-filled roles was a natural one, given the distinction between functions and relations in first-order logic, and the common use of single-valued relations in feature logics and relational databases.[7]

Other aspects of CLASSIC evolved for similar reasons. For example, while our original specification for a **TEST** concept was conceptually sufficient, we left it

[7] Attributes correspond to functionally-dependent columns in relations, whereas multiply-filled roles would most easily correspond to two-column relations. These correspondences turned out to be of great use to us when we subsequently attempted to interface CLASSIC to a relational database [14].

to the user to specify (optionally) a "realm" (i.e., *CLASSIC* or *HOST*). This made the processing more complex—and different—for concepts that had **TEST**s than for those that did not. It was also the only case of a concept construct for which the user had to specify a realm at all. In order to make the code more simple and reliable, and the interface more uniform, we eventually substituted for **TEST** two different constructors (**TEST-C**, **TEST-H**), which would each be unambiguous about its realm.

Implementation Tradeoffs. It is well known that creating computer programs involves making tradeoffs between time and the use of space. In most cases, decisions made because of efficiency should not be of consequence to the system's functional interface. However, some tradeoffs can be extremely consequential, and yet never occur to designers of an unimplemented language.

One tradeoff that affected our user language involved the form and utility of role inverses. If one could afford to keep backpointers from every filler back to every role it fills, then one could have an (**INVERSE** <role>) construct appear any place in the language that a role can. This would be quite convenient for the user and would contribute to the uniformity of the language—but it would also entail significant overhead. An alternate view is to force users to explicitly declare in advance any role inverses that they intend to use at the same time the primitive roles are declared. Then, backpointers would be maintained only for explicitly declared roles. The point here is not which approach is best, but rather that practical considerations can have significant effects on a pure language that never takes such things into account. Given the large difference between the two approaches, and inferential difficulty that results from including inverses in the language, we chose to exclude them altogether from the first release. A more recent version of CLASSIC included them, since we subsequently had a chance to think hard about the interplay between language, operation, and implementation design.

While the original specification of CLASSIC did not account for *retraction* of information, our applications soon forced us into providing such a facility. In this case, retraction became one of the key reflectors of implementation tradeoffs. Most reasonable implementations of retraction in inference systems keep track of dependencies. Given extremely large knowledge bases, it may be too expensive (both in space and time) to keep track of such dependencies at a very fine-grained level of detail. Because of this, CLASSIC has a unique medium-grain-sized dependency mechanism, such that for each individual, all other individuals can be found where a change to one of them could imply a change to the original. This medium-grained approach saves space over approaches (e.g., [24]) that keep track of dependencies at the assertion level, or approaches that keep track of all dependencies in a truth-maintenance fashion. The reduction in the amount of record-keeping also saves time, which we believe even results in an overall faster system.

4.3 Serving the General User Population

Real use of a system, regardless of the particular applications supported, immediately makes rigorous demands on a formalism that may otherwise look good on paper. We consider three types of issues here: (1) comprehension and usability of the language by users; (2) specific features of the user interface, e.g., error-handling and explanation, that make the system more usable; and (3) getting around gaps in the logic.

Usability of the Language. Concepts like "usability" are admittedly vague, but it is clear that users will not stick with a system if the abstractions behind its logic and interface do not make sense. Formal semantics makes precise what things mean, and it behooves us to provide such formal bases for our logics. However, how simple a language is to learn and how easy it is to mentally generate the name of a function that is needed are more likely the *real* dictators of ultimate success or failure.

In the CLASSIC world, this meant (among other things) that the language should be as uniform as possible—the more special cases, the more problems. Just being forced to think about this led us to an insight that made the language better: there was in general no good reason to distinguish between what one could say about an individual and what one could use as part of a concept. (Note in the grammar of Figure 1 that concept-expressions and individual-expressions are treated differently.) The **FILLS** constructor should have been equally applicable to both; in other words, it makes sense to form the general concept of "an automobile whose manufacturer is Volvo," where Volvo is an individual:

```
(AND AUTOMOBILE (FILLS manufacturer Volvo))
```

In the original specification, we thought of role-filling as something one does exclusively in individuals. The one sticking point to a generalization was the **CLOSE** constructor, which we felt did not make much sense for concepts; but as we see below, further thinking about **CLOSE** (instigated by user concerns) eventually led us to determine that it was mistakenly in the language in the first place. As a result, the types of descriptions allowable as definitions of concepts and for assertions about individuals could be merged.

There were other simplifications based on generic user concerns like understandability that helped us derive an even cleaner logic. For example, the **PRIMITIVE** and **DISJOINT-PRIMITIVE** concept-forming constructors,

which had a firm semantics but were found problematic by non-experts in actual use, were removed from the language and better instantiated as variants on the concept-defining interface function. The conceptually adequate but awkward arguments to **DISJOINT-PRIMITIVE** were also simplified.

While we provided all of our research papers, potential users demanded usage guidelines aimed at non-PhD researchers, to aid their comprehension of the logic. In an effort to educate people on when a description logic-based system might be useful, what its limitations were, and how one might go about using one in a simple application, a long paper was written with a running (executable) example on how to use the system [12]. This paper discussed typical knowledge bases, useful "tricks of the trade," ideas in our logic that would be difficult for non-KR people, and a conventional methodology for building CLASSIC KB's.

Motivated by the need to help users understand CLASSIC's reasoning paradigm and by the need to have a quick prototyping environment for showing off novel functionality, we developed several demonstration systems. The first such system was a simple application that captured "typical" reasoning patterns in an accessible domain—advising the selection of wines with meals. While this application was appropriate for many students, an application more closely resembling commercial applications in configuration was needed to give more meaningful demonstrations internally and to provide concrete suggestions of new functionality that developers might consider using in their applications. This led to a more complex application concerning stereo system configuration, which had a fairly elaborate graphical interface [29,28]. Both of these applications have subsequently been adapted for the Web.

Motivated by the need to grow a larger community of people trained in knowledge representation in general and description logics in particular, we collaborated with a corporate training center to generate a course. Independently, at least one university developed a similar course and a set of five running assignments to help students gain experience using the system. We collaborated with University of Pittsburgh on the tutorial to support the educators and to gather feedback from the students. The student feedback from yearly course offerings drove many of our environmental enhancements such as enhanced explanation support for contradictions, pruning, and debugging.

All of this effort in building user aids seemed truly "ancillary" at the beginning, but proved to be crucial in the end.

Human Interface Features. Even with a perfectly understandable and intuitive logic, a minimal, raw implementation will be almost impossible to use. In general, our customers told us, the system's development environment (for building and debugging knowledge bases) was a make-or-break concern. For example, logics discussed in papers do not deal with issues like *error-handling*, yet real users can not use systems unless they get meaningful error-reporting and reasonable error-handling, especially when the KR system is embedded in a larger system. As a result of direct and strong feedback from users, the released version of CLASSIC had extensive error-handling, including well-documented return codes and rational and consistent error returns.

More specifically, our configuration applications relied heavily on the detection of *contradictions*, since users would, among other things, try out potential updates in a "what-if" mode. Certain input specifications might lead to an inconsistency with the updates that had previously been made. One of the key aspects of contradiction-handling, then, was the need to roll back the effects of the update that caused such an inconsistency in the knowledge base. Since CLASSIC was able to do elaborate propagation and rule-application, a long and ramified inference chain may have been triggered before a contradiction was encountered, and unless every piece of that chain were removed, the knowledge base would be left in an incoherent state. This need led us to consider ways to unravel inferences, including the possible use of a database-style "commit" operation (i.e., the knowledge base would never be changed until all inferences concluded successfully).

We eventually settled on a more conventional AI approach using dependencies, which gave us a general facility that not only would guarantee the KB to be returned to a meaningful state after a contradiction occurred, but would allow the user direct *retraction* of facts previously told. As it turned out, the availability of such a retraction capability was critical in "selling" the application to its sponsors, since the ability to explore alternative options in unconstrained ways was essential to the interactive customer sales process.

Another generic area that needed attention was *explanation* of reasoning—a topic relatively ignored by the KR community. If users are to build nontrivial KB's, they will need help in understanding and debugging them; they will need to know why an inference failed, or why a conclusion was reached. While the expert systems community may have learned this lesson, it is an important one for those working in general KR as well. Our users made a very strong case to us that such a feature was critical to their successful construction of knowledge bases.

We responded by adding an explanation mechanism to CLASSIC [25,27]. Since the key inference in CLASSIC is subsumption, its explanation forms the foundation of an explanation module. Although subsumption is calculated procedurally in CLASSIC, we found it necessary to provide a declarative presentation of CLASSIC's deductions in order to reduce the length of explanations and to remove the artifacts of the procedural implementation. We used an incremental proof-theoretic foundation and applied it to all of the inferences in CLASSIC, including the inferences for handling constraint propagation and other individual inferences. This basic explanation foundation has proved useful and general and since then has been used (in joint work with Ian Horrocks and Enrico Franconi) as the foundation for a design for explaining the reasoning in tableaux-based description logic reasoners, and also (in joint work with James Rice) in an implemented system for explaining the reasoning in a model-elimination theorem prover at Stanford.

As soon as we had both explanation and appropriate handling of contradictions in CLASSIC, we found that specialized support for explanation of contradictions was called for. If an explanation system is already implemented, then explaining contradictions is *almost* a special case of explaining any inference, but with a twist. Information added to one object in the knowledge base may cause another object to become inconsistent. Typical description logic systems, including CLASSIC, require consistent knowledge bases, thus whenever they discover a contradiction, they use some form of truth maintenance to revert to a consistent state of knowledge (as mentioned above), removing conclusions that depend on the information removed from the knowledge base. But a simple-minded explanation based solely on information that is currently in the knowledge base would not be able to refer to these removed conclusions. Thus, any explanation system capable of explaining contradictions would need to access its inconsistent states as well as the current state of the knowledge base.

Another issue relevant to explanation is the potential incompleteness of the reasoner. In particular, a user might have an intuition that some conclusion should have been reached, but the system did not reach it. To explain this might in general require using a different, complete reasoner, but frequently occuring special cases can be built into the system itself.[8]

As CLASSIC makes it easy to generate and reason with complicated objects, our users found naive object presentations to be overwhelming. For example, in our stereo demonstration application, a typical stereo system description generated four pages of printout. This contained clearly meaningful information, such as price ranges and model numbers, but also descriptions of where the component might be displayed in the rack and which superconcepts were related to the object. In fact, in some contexts it might be desirable to print just model numbers, while in other contexts it might be desirable to print price ranges and model numbers of components.

To reduce the amount of information presented in CLASSIC explanations we added facilities for describing what is interesting to print or explain on a concept-by-concept basis. This led us to a meta-language for matching "interesting" aspects of descriptions [25,8]. The approach provides support for encoding both domain-independent and domain-dependent information to be used along with context to determine what information to print or explain. The meta-language essentially extends the base description logic with some carefully chosen auto-epistemic constructors ("Is at least one filler known?") to help decide what to print. As a result, in one application object presentations and explanations were reduced by an order of magnitude, which was essential in making the project practical.

Overcoming Gaps in the Logic. Another key point of tension between theory and practice is the notion of an *"escape"* for the user, e.g., a means to get around an expressive limitation in the logic by resorting to raw LISP or C code. As mentioned above, we included in the original specification a **TEST** construct, which allowed the user to resort to code to express sufficiency conditions for a concept. In the original paper, we did not technically include **TEST**s in concept definitions, since no formal semantics was available for it. We quickly provided guidelines (e.g., avoiding side-effects) that could guarantee that **TEST**-defined concepts could fit into our formal semantics, even if the **TEST** code itself was opaque to CLASSIC. But our view was that the **TEST** construct was not intended to be a general programming interface.

As it turned out, **TEST**-concepts were one of the absolute keys to successful use of CLASSIC. In fact, they not only turned out to be a critical feature to our users, but as we observed the patterns of tests that were written in real applications, we were able to ascertain a small number of new features that were missing from the language but fundamental to our applications. First, we discovered that users consistently used **TEST**s to encode simple numerical range restrictions; as we mention below, this led us to create **MAX** and **MIN** constructors for our concept language. Later, in one significantly large and real-world application,

[8] In the case of CLASSIC, inferences not supported by the modified semantics of individuals used in subsumption reasoning fall into this category.

we found only six different patterns of **TEST** concepts, with over 85% of these falling into only two types; one was computing via a function a universal restriction on a role (actually, a numerical range), and the other was computing a numerical filler for a role (a simple sum). We have subsequently made additions to CLASSIC to accommodate these common patterns of usage (i.e., "computed rules"), and have found that newer versions of the same knowledge base are substantially simpler, and less prone to error (the original **TEST**s were written to achieve some of their effects by side-effect, which we subsequently eliminated).

Thus, while our original fear was that an escape to LISP or C was an embarrassing concession to implementation, and one that would destroy the semantics of the logic if used, our **TEST**s were never used for arbitrary, destructive computation. Rather, this mechanism turned out to be a means for us to measure specifically where our original design was falling short, all the while staying within a reasonable formal semantics.[9]

4.4 Meeting the Needs of Particular Applications

As soon as a system is put to any real use, mismatches or inadequacies in support of particular applications become very evident. In this respect, there seems to be all the difference in the world between the few small examples given in research papers and the details of real, sizable knowledge bases. As mentioned, we took on several significant and different types of applications. While the demands from each of them were somewhat different, they clearly did *not* demand that we immediately extend CLASSIC to handle the expressive power of full first-order logic. In fact, the limited number of extensions and changes that arose from the interaction with individual applications are useful in all of them, and all stay within the original spirit of simplicity.

Language and KB Operation Features. Among the first needs we had to address was the significance of numbers and strings. Virtually all of the applications needed to express concepts limiting the values of roles that had *HOST* values in them, as in, for example, "a manager whose salary is between 20000 and 30000." On the one hand, this need vindicated our original decision to integrate host information in a serious manner.[10] On the other, as mentioned above, the need to create **TEST**-concepts just to test simple ranges like this showed us that we would have a hard time measuring up to almost any application that

[9] As evidence of the continuing general lack of appreciation of the more theoretically-inclined towards pragmatic issues, consider that one of the reviewers of our 1989 paper [5] called **TEST**s "an abomination." Yet, not only were they undeniably critical to our users, we managed to keep them in line semantically and they provided concrete input concerning the expressive extensions that were most required by our users.

[10] We should point out that integration here is not just a simple matter of allowing numbers or strings in roles; it has ramifications for the language syntax and parsing, part of the concept hierarchy must be built automatically, data structures for *CLASSIC* individuals need to be carefully distinguished from arbitrary LISP structures, etc.

used real data (especially if it came from a DBMS). Thus, recent versions of CLASSIC have new concept types that represent ranges of *HOST* values.[11] These are integrated in a uniform manner with other concept constructors, and the semantics accounts for them.

Another major consequence of dealing with a significant application is the reality of *querying* the KB. Our original design of CLASSIC (as was the case with virtually all frame systems) paid scant attention to queries other than those of the obvious sort, e.g., retrieving instances of concepts. Once we began to see CLASSIC as a kind of deductive database manager, we were forced to face the problem that our querying facilities were very weak. This led to the design of a substantial query language for CLASSIC, which could handle the needed object-oriented queries, as well as the SQL-style retrievals that are so common in the real world of information management. While this is not profound (although the query language we developed has some novel features and is itself an important contribution), the key point is that it was the attempt at application that made us realize that an entire critical component was missing from our work.

Two other consequences of this sort bear brief mention.

First, our simple notion of a **TEST** was sufficient to get us off the ground. Our intention was to pass the individual being tested to the test function as a single argument. As it turned out, our users needed to pass other items in as arguments. For example, if the test were a simple function to compute whether the value of a role were greater than some number, say 5, then the number 5 should have been coded directly into the test function; this, in turn, would have led to the creation of many almost-identical functions—unless we provided the ability to pass in additional arguments. We have done so in the latest versions of CLASSIC.

Second, our original design of rules was a sufficient foundation, but it required a named concept to exist as the left-hand-side of the rule. As soon as some of our users tried to use this, they found that they had to construct concepts artificially, just to serve to invoke the rules. While this posed no conceptual problem for the logic, and no semantic aberration, it became a practical nightmare. Thus, it was important to extend our rules to allow a filter; in other words, the rule could be associated with the most general named concept for which it made sense, but only fired when a filtering subcondition was satisfied. This now avoids needless creation of artificial concepts.

API's. Finally, an important consideration was the relationship between our KR system and the application that used it. In the vast majority of our applications, CLASSIC had to serve as a tightly integrated component of a much larger overall system. For this to be workable, CLASSIC had to provide a full-featured application programming interface (API) for use by the rest of the system.

[11] **MAX** and **MIN** have instances that are numbers; e.g., (**MAX** 25) represents the set of integers that are less than or equal to 25. These are used to restrict the *value* of a filler of a role; for example, we could use **MAX** to specify the value restriction on a person's age, as in (**AND** PERSON (**ALL** age (**MAX** 25))). **AT-LEAST** and **AT-MOST**, on the other hand, restrict the *number of fillers* of a role, not their values.

Our most complete API was in the NeoClassic (C++) system. It had the usual calls to add and retract knowledge and to query for the presence of particular knowledge. In addition, there was a broader interface that let the rest of the system receive and process the data structures used inside NeoClassic to represent knowledge, but without allowing these structures to be modified outside of NeoClassic.[12] This interface allowed for much faster access to the knowledge stored by NeoClassic, as many accesses were simply to retrieve fields from a data structure. Further, direct access to data structures allowed the rest of the system to keep track of knowledge from NeoClassic without having to keep track of a "name" for the knowledge, and also supported explanation.

A less-traditional interface that is provided by both LISP CLASSIC and NeoClassic is a notification mechanism ("hooks"). This mechanism allows programmers to write functions that are called when particular changes are made in the knowledge stored in the system or when the system infers new knowledge from other knowledge. Hooks for the retraction of knowledge from the system are also provided. These hooks allow, among other things, the creation of a graphical user interface that mirrors (some portion or view of) the knowledge stored in the representation system.

Lately, others in the knowledge representation community have recognized the need for common API's, (e.g., the general frame protocol [17] and the open knowledge base connectivity [16]) and translators exist between the general frame protocol API specification and CLASSIC.

4.5 Revisiting What Looked Good on Paper

Probably more commonly than researchers would like to admit, theoretical KR papers are not always what they seem. While theorems and formal semantics help us get a handle on the consequences of our formalisms, they do not always do a complete job; it is also not unheard of for them to contain mistakes. Our own experience was that several parts of our original formalism were clarified substantially by the experience of having to implement an inference algorithm and have it used on real problems. In each case, a change was necessary to the original formal work to accommodate the new findings. Because of the complexities and subtleties of real-world problems, and the extreme difficulty of anticipating in the abstract what real users will want, it seems that this type of effect is inevitable, and a critical contribution of practice over pure "theory."

For example, we had originally proposed that **CLOSE** could appear in a description applied to an individual, to signal that the role fillers asserted by the description were the only fillers. Thus, one could assert of Dale,

```
(AND (FILLS friend Audrey)
     (FILLS friend Harry)
     (CLOSE friend));
```

[12] Of course, as C++ does not have an inviolable type system, there are mechanisms to modify these structures. It is just that any well-typed access cannot.

this was to mean that Audrey and Harry were Dale's *only* friends. We thought of this, semantically, as a simple predicate closure operation. However, once a real knowledge base was constructed, and users started interacting with the system, we discovered a subtle ordering dependency: pairs of **CLOSE** constructors could produce different effects if their order were reversed; this occured because the first closing could trigger a rule firing, whose effect could then be to enable or block another rule firing in conjunction with the second closing. This led us to discover that our original characterization of **CLOSE** was in general wrong. In reality, it had an autoepistemic aspect, and thus closing roles had to become an operation on an entire knowledge base, and could not be part of a larger expression. **CLOSE** was thus removed from the description language and made a knowledge base operation.

We had a small number of similar experiences with other aspects of the language. For instance, our original estimation was that a certain part of CLASSIC's reasoning with individuals was complete for the purposes of subsumption checking. In implementation, we had to look substantially closer at what properties of individuals could count in the subsumption of concepts (individuals could appear in concepts originally in **ONE-OF** constructs, and later, in **FILLS** expressions). In doing so, we discovered that while the implementation actually did what we thought was right—it ignored contingent properties of individuals in subsumption calculations—the semantics was wrong. We eventually found a suitable, interesting, and somewhat non-standard semantic account that described what we *really* meant [7]. Moreover, it was discovered ([40] and, independently, [7]) that reasoning with **ONE-OF** according to standard semantics is intractable. In retrospect, our belief is that it would have been a mistake to omit individuals from concept descriptions because of this complexity result, and that our intuitions serendipitously led us to a good compromise. So, while formal semantics are a good foundation on which to build, they are not necessarily what the designers mean or what the users need to understand.

Being forced by implementation to get every last detail right also caused us ultimately to better understand our **TEST** constructs. Given when they would be invoked, it eventually became clear (thanks to a key user's discovery) that **TEST**s, which were originally two-valued functions, had to be *three*-valued: since CLASSIC supports partial information about individuals, it is possible for a test to "fail" at one point and succeed later, even with strictly monotonic additions to the KB. If the test truly failed the first time, and the individual were determined *not* to satisfy a description based on this failure, nonmonotonicity would be introduced in an inappropriate way. Thus **TEST** functions need to tell their caller if the individual provably satisfies the test, provably fails it, or neither.

4.6 Other Important Influences

While our focus here has been on the feedback from our practice with CLASSIC to our theory, it is important to point out that our practical work on CLASSIC both spawned and benefited from other theoretical work.

For example, we had not worried about that fact that expanding the definitions of concepts could, in the worst case, lead to an exponential blow-up in the space required to represent a CLASSIC knowledge base, but we did not know whether there was perhaps a better method. Work by Nebel [34] showing that there is an inherent intractability in the processing of description logic knowledge bases made the existence of such a method unlikely.

As mentioned, the original **CLOSE** constructor had to be abandoned because of implementation ordering difficulties. We replaced the **CLOSE** constructor with an operation on knowledge bases, to which we gave an operational semantics. Donini, *et al.*, [20] realized the true epistemic nature of this operation, and pointed out that the trigger rules used in CLASSIC and LOOM also have such an epistemic nature. As a result, a theory of epistemic DL's was developed, where rules like `PERSON` $\rightsquigarrow$`(`**ALL** `parents PERSON)`, hitherto given an operational interpretation, were integrated into the knowledge base by using a modal operator **K**: **K**`(PERSON)` $\Longrightarrow$ `(`**ALL** `parents PERSON)`, and thus given a denotational semantics. This semantics provides a justification for the operational treatment provided in CLASSIC.

The successful implementation and use of filtering for elimination of uninteresting information about individuals led to its formalization through the notion of *patterns*—descriptions with variables occurring in some places instead of identifiers—and pattern matching [8]. This formal work has been extended to other languages [3], and may have applications to knowledge-base integration, where concepts from one ontology may be matched against corresponding parts of the other ontology, in order to discover commonalities [6].

Inspired by the need for both domain-independent extensions (e.g., qualified number restriction), and domain-specific ones (e.g., reasoning with dates, plans, etc.), the CLASSIC implementation was analyzed, rationalized and generalized to an architecture that supports the addition of new concept constructors [4]. In fact, the last version of LISP CLASSIC (release 2.3) has features for adding subsumption reasoning for some **TEST**-concepts, because a description like `(`**TEST-H** `initialSubstring "reason")`—denoting strings that begin with the letters `r-e-a-s-o-n`—can be viewed as just a syntactic variant of the description `(`**INITIAL-SUBSTRING** `"reason")`, which makes it clear that a new constructor is being used. Note that the addition of arguments to **TEST**-concept functions was crucial in this step to extensibility.

5 Modern CLASSIC

The result of the long and arduous trail implied above, from typical research paper to practical system, was a significant improvement in the CLASSIC language and the clarity of operations on a CLASSIC knowledge base. The basic expression grammar was simplified and made more uniform (see Figure 3), and the semantics was adjusted to be truer to our original intention. KB operations were streamlined and made more useful, and error-handling and retraction were added. The resulting system is unarguably superior to the original in every way:

```
<concept-expression> ::=
        THING | CLASSIC-THING | HOST-THING | NUMBER | STRING |
        <concept-name> |
        (AND <concept-expression>+) |
        (ALL <role-name><concept-expression>) |
        (AT-LEAST <positive-integer><role-name>) |
        (AT-MOST <non-negative-integer><role-name>) |
        (FILLS <role-name> <individual-name>+) |             % added for uniformity
        (SAME-AS (<attribute-name>+) (<attribute-name>+)) |          % restricted
        (TEST-C <function><arg>*) |          % clarified; arguments added; 3-valued
        (TEST-H <function><arg>*) |          % clarified; arguments added; 3-valued
        (ONE-OF <individual-name>+) |
        (MAX <number>) |                                                 % added
        (MIN <number>)                                                   % added
<individual-expression> ::=
        <concept-expression> |                      % made uniform with concepts
        <individual-name> |
        <host-language constant>
```

Fig. 3. The Resulting CLASSIC Concept Language

it has new constructs that meet real needs, substantial parts of it have been validated by use, the overall interface makes more sense, it is cleaner and more elegant, and it is devoid of flaws that were subtly hidden in the original.

The effects of the pragmatic factors we have described here are varied, and not easily classified. But they are clearly substantial and were critical to the success and ultimate form of CLASSIC. To summarize, here are some of the most important changes that were driven by the attempt to "reduce" the system to practice and put it to the test of real use:

– *language improvements:* equal descriptive power for individuals and concepts; distinction between attributes and multiply-filled roles; **SAME-AS** applicable to attributes only and efficiently computable; arguments for **TEST**-concepts; three-valued **TEST**s; completely compositional language with no order dependencies; numeric range concepts; rules with filter conditions, no longer requiring artificial concepts; realms of **TEST**-concepts unambiguous and **TEST** constructs made uniform with other parts of language; computed rules;
– *interface improvements:* primitive and disjoint primitive definition as KB operators; disjoint primitive specification simplified; **CLOSE** as a KB operator; sophisticated query language and implemented query processor; complete API for embedded use;
– *system features:* comprehensive error-reporting and handling; extensive explanation capabilities; filtering language for pruning; renaming of concepts; retraction of "told" information; contradiction-handling.

Finally, we completed the cycle by embarking on an actual formal proof of the tractability of CLASSIC and the completeness of our reasoner [7]. This proof was more difficult than usual because the language lacks negation, so standard techniques could not be applied. We ended up using an abstraction of the implementation data structure for the proof, and we must admit that it took the trained eye of a very devoted and skilled reviewer to get the details right. So, while it would have been nice to have come up with this proof before we even proposed the logic and tried to implement it, it is very doubtful that we would have succeeded, without the experience of practice to guide us.

6 Lessons

The main lesson to be learned here is that despite the ability to publish theoretical accounts of logics and their properties, the true theoretical work on KR systems is not really done until issues of implementation and use are addressed head-on. The basic ideas can hold up reasonably well in the transition from paper to system, but traditional research papers miss many make-or-break issues that determine a proposal's true value in the end. Arguments about needed expressive power, the impact of complexity results, the naturalness and utility of language constructs, etc., are all relatively hollow until made concrete with specific applications and implementation considerations.

Although a complete formal specification of a knowledge representation system (including an algorithmic specification of the inferences that the system is required to perform and a computational analysis of these inferences) is essential, the presence of a formal account is not sufficient for the success of the system. There is no guarantee, for example, that a formally tractable knowledge representation system can be effectively implemented, as it may be exceedingly difficult to code the inference algorithms or other portions of the system efficiently enough for use or perspicuously enough to tell if they are correct. Even then, there is no guarantee that the resulting system will be useful in practice, even if it appears at first glance to meet some apparent needs. Finally, getting the formal specification *really* right is an extremely difficult task, especially for systems that perform partial reasoning or which have non-standard but useful constructs. All told, *the implementation and use of the system is a vital complement to work on knowledge representation "theory."* It can illuminate problems in the formal specification, and will inevitably provide real problems for the theory side to explain.

Our experience with CLASSIC has taught us this lesson in some very specific ways. Any hope of having the system make a real impact (e.g., in a product) rested on some very practical considerations that in some cases were impossible to anticipate before interacting with developers. We learned through extensive interaction with our developers that issues like upward compatibility and simplicity were in some ways much more important than individual features. We learned that usability issues such as explanation, contradiction-handling, and pruning were critical to longevity and maintenance in applications and were much more important than additional language constructs. We learned that attention to complexity (although not maniacal concern with it) was very much worth the effort, because of the critical impact of performance and predictability on acceptance of the system. We also learned that we could not afford to be totally rigid on any point—be it language features, complexity, or names of functions—without jeopardizing potential use of the system. The feeling of the CLASSIC group is that the resulting system is clearly far better than anything we could have built in a research vacuum. And the effort of reducing our ideas to a practical system generated a great deal of research—on language constructs, complexity, and even formal semantics—that was not only interesting, but important simply by virtue of the very fact that it arose out of real problems.

At a more strategic level, one very important lesson for us was the significance of a certain kind of conservatism. We could have invested a large amount of time designing features and providing expressive power (and implementation complexity) that would have, as it turned out, gone completely to waste. On the flip side, our users gave us clear and direct evidence of features that they did need, and that we were not providing, via our **TEST** construct, which, to be honest, surprised us both in its criticality and in the simple and regular ways in which it was used—not to mention the smallness of the number of needed extensions. All told, our decision to start with a small (but not hopelessly impoverished) language, with room for growth in a reasoned fashion, was clearly a successful one. While such emphasis on simplicity might not necessarily be right for all projects, given the constraints under which product developers live, it is a key issue to consider when the practice that we are "reducing" to is not just for AI research but for development and product.

In sum, a number of key factors of a strongly pragmatic sort show that logics that look good on paper may have a long way to go before they can have any impact in the real world. These factors range from upward compatibility and system maintenance to implementation tradeoffs and critical system features like error-handling and explanation. They include learnability of the language and occasional escapes to circumvent limitations of the system. While individual gains in CLASSIC derived from attention to these practical concerns may each have been small, they all added up, and made a big difference to success of the system as a whole. It is quite clear that if practical concerns were ignored, the resulting system would have had at best limited utility. In fact, in general in our field, it seems that the true theoretical work is not done until the implementation runs and the users have had their say.

References

1. Aït-Kaci, H.: Type subsumption as a model of computation. In: Kerschberg, L. (ed.) Proceedings of the First International Conference on Expert Database Systems, Kiawah Island, South Carolina, October 1984, pp. 124–150 (1984)
2. Baader, F., Hollunder, B., Nebel, B., Profitlich, H.-J., Franconi, E.: An empirical analysis of optimization techniques for terminological representation systems, or, making KRIS get a move on. In: Nebel, et al. (eds.) [35], pp. 270–281
3. Baader, F., Küsters, R., Borgida, A., McGuinness, D.: Matching in description logics. Journal of Logic and Computation 9(3), 411–447 (1999)

4. Borgida, A.: Extensible knowledge representation: the case of description reasoners. Journal of Artificial Intelligence Research 10, 399–434 (1999)
5. Borgida, A., Brachman, R.J., McGuinness, D.L., Resnick, L.A.: CLASSIC: A structural data model for objects. In: Proceedings of the 1989 ACM SIGMOD International Conference on Mangement of Data, June 1989, pp. 59–67. Association for Computing Machinery, New York (1989)
6. Borgida, A., Küsters, R.: What's NOT in a name?: Initial explorations of a structural approach to intgerating large concept knowledge bases. Technical Report DCS-TR-391, Rutgers University, Dept. of Computer Science (August 1999)
7. Borgida, A., Patel-Schneider, P.F.: A semantics and complete algorithm for subsumption in the Classic description logic. Journal of Artificial Intelligence Research 1, 277–308 (1994)
8. Borgida, A., McGuinness, D.L.: Inquiring about frames. In: Aiello, L.C., Doyle, J., Shapiro, S.C. (eds.) Principles of Knowledge Representation and Reasoning: Proceedings of the Fifth International Conference (KR 1996), November 1996, pp. 340–349. Morgan Kaufmann Publishers, San Francisco (1996)
9. Brachman, R.J.: A Structural Paradigm for Representing Knowledge. PhD thesis, Harvard University, Cambridge, MA (1977); BBN Report No. 3605, Bolt Beranek and Newman, Inc., Cambridge, MA (July 1978) (revised version)
10. Brachman, R.J., Fikes, R.E., Levesque, H.J.: KRYPTON: Integrating terminology and assertion. In: Proceedings of the Third National Conference on Artificial Intelligence, Washington, DC, August 1983, pp. 31–35. American Association for Artificial Intelligence, Menlo Park (1983)
11. Brachman, R.J., Levesque, H.J.: The tractability of subsumption in frame-based description languages. In: Proceedings of the Fourth National Conference on Artificial Intelligence, Austin, Texas, August 1984, pp. 34–37. American Association for Artificial Intelligence, Menlo Park (1984)
12. Brachman, R.J., McGuinness, D.L., Patel-Schneider, P.F., Resnick, L.A., Borgida, A.: Living with CLASSIC: When and how to use a KL-ONE-like language. In: Sowa, J.F. (ed.) Principles of Semantic Networks: Explorations in the representation of knowledge, pp. 401–456. Morgan Kaufmann Publishers, San Francisco (1991)
13. Brachman, R.J., Schmolze, J.G.: An overview of the KL-ONE knowledge representation system. Cognitive Science 9(2), 171–216 (1985)
14. Brachman, R.J., Selfridge, P.G., Terveen, L.G., Altman, B., Borgida, A., Halper, F., Kirk, T., Lazar, A., McGuinness, D.L., Resnick, L.A.: Knowledge representation support for data archaeology. In: First International Conference on Information and Knowledge Management, Baltimore, MD, November 1992, pp. 457–464 (1992)
15. Bresciani, P., Franconi, E., Tessaris, S.: Implementing and testing expressive description logics: a preliminary report. In: Ellis, G., Levinson, R.A., Fall, A., Dahl, V. (eds.) Knowledge Retrieval, Use and Storage for Efficiency: Proceedings of the First International KRUSE Symposium, pp. 28–39 (1995)
16. Chaudhri, V.K., Farquhar, A., Fikes, R., Karp, P.D.: Open Knowledge Base Connectivity 2.0. Technical report, Technical Report KSL-09-06, Stanford University KSL (1998)
17. Chaudhri, V.K., Farquhar, A., Fikes, R., Karp, P.D., Rice, J.: The Generic Frame Protocol 2.0. Technical report, Artificial Intelligence Center, SRI International, Menlo Park, CA (July 1997)
18. Devanbu, P., Brachman, R.J., Ballard, B., Selfridge, P.G.: LaSSIE: A knowledge-based software information system. Communications of the ACM 34(5), 35–49 (1991)

19. Donini, F.M., Lenzerini, M., Nardi, D., Nutt, W.: Tractable concept languages. In: Proceedings of the Twelfth International Joint Conference on Artificial Intelligence. International Joint Committee on Artificial Intelligence, Sydney, Australia, August 1991, pp. 458–453 (1991)
20. Donini, F.M., Lenzerini, M., Nardi, D., Schaerf, A., Nutt, W.: Adding epistemic operators to concept languages. In: Nebel et al. (ed.) [35], pp. 342–353
21. Doyle, J., Patil, R.: Two theses of knowledge representation: Language restrictions, taxonomic classification, and the utility of representation services. Artificial Intelligence 48(3), 261–297 (1991)
22. Horrocks, I.: Using an expressive description logic: FaCT or fiction? In: Cohn, A.G., Schubert, L., Shapiro, S.C. (eds.) Principles of Knowledge Representation and Reasoning: Proceedings of the Sixth International Conference (KR 1998), June 1998, pp. 636–647. Morgan Kaufmann Publishers, San Francisco (1998)
23. International Joint Committee on Artificial Intelligence. In: Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (August 1995)
24. MacGregor, R.M.: A deductive pattern matcher. In: Proceedings of the Seventh National Conference on Artificial Intelligence, St. Paul, Minnesota, August 1988, pp. 403–408. American Association for Artificial Intelligence, Menlo Park (1988)
25. McGuinness, D.L.: Explaining Reasoning in Description Logics. PhD thesis, Department of Computer Science, Rutgers University (October 1996); also available as Rutgers Technical Report Number LCSR-TR-277
26. McGuinness, D.L.: Ontological issues for knowledge-enhanced search. In: Proceedings of Formal Ontology in Information Systems. IOS-Press, Washington (1998); also In: Frontiers in Artificial Intelligence and Applications (to appear)
27. McGuinness, D.L., Borgida, A.: Explaining subsumption in Description Logics. In: IJCAI 1995 [23], pp. 816–821
28. McGuinness, D.L., Patel-Schneider, P.F., Resnick, L.A., Isbell, C., Parker, M., Welty, C.: A description logic-based configuration for the web. SIGART Bulletin 9(2) (fall, 1998)
29. McGuinness, D.L., Resnick, L.A., Isbell, C.: Description Logic in practice: A CLASSIC application. In: IJCAI-1995 [23], pp. 2045–2046
30. McGuinness, D.L., Wright, J.R.: Conceptual modeling for configuration: A description logic-based configurator platform. Artificial Intelligence for Engineering Design, Analysis, and Manufacturing Journal - Special Issue on Configuration (1998)
31. McGuinness, D.L., Wright, J.R.: An industrial strength description logic-based configuration platform. IEEE Intelligent Systems (1998)
32. Moser, M.G.: An overview of NIKL, the new implementation of KL-ONE. Technical Report 5421, BBN Laboratories, 1983. Part of a collection entitled "Research in Knowledge Representation for Natural Language Understanding—Annual Report (September 1, 1982–August 31, 1983)
33. Mylopoulos, J., Bernstein, P., Wong, H.K.T.: A language facility for designing database-intensive applications. ACM Transactions on Database Systems 5(2), 185–207 (1980)
34. Nebel, B.: Terminological reasoning is inherently intractable. Artificial Intelligence 43(2), 235–249 (1990)
35. Nebel, B., Rich, C., Swartout, W. (eds.): Principles of Knowledge Representation and Reasoning: Proceedings of the Third International Conference (KR 1992). Morgan Kaufmann Publishers, San Francisco (1992)

36. Patel-Schneider, P.F.: Small can be beautiful in knowledge representation. In: Proceedings of the IEEE Workshop on Principles of Knowledge-Based Systems, Denver, Colorado, December 1984, pp. 11–16. IEEE Computer Society, Los Alamitos (1984)
37. Patel-Schneider, P.F.: Undecidability of subsumption in NIKL. Artificial Intelligence 39(2), 263–272 (1989)
38. Patel-Schneider, P.F.: DLP system description. In: Franconi, E., De Giacomo, G., MacGregor, R.M., Nutt, W., Welty, C.A. (eds.) Proceedings of the 1998 International Workshop on Description Logics, June 1998, pp. 87–89 (1998); available electronically as a CEUR publication at `http://SunSite.Informatik.RWTH-Aachen.DE/Publications/CEUR-WS/Vol-11/`
39. Peltason, C., von Luck, K., Nebel, B., Schmiedel, A.: The user's guide to the BACK system. KIT-Report 42, Fachbereich Informatik, Technische Universität Berlin (January 1987)
40. Schaerf, A.: Reasoning with individuals in concept languages. Data and Knowledge Engineering 13(2), 141–176 (1994)
41. Schmidt-Schauss, M.: Subsumption in KL-ONE is undecidable. In: Brachman, R.J., Levesque, H.J., Reiter, R. (eds.) Principles of Knowledge Representation and Reasoning: Proceedings of the First International Conference (KR 1989), May 1989, pp. 421–431. Morgan Kaufmann Publishers, San Francisco (1989)
42. Selfridge, P.: Knowledge representation support for a software information system. In: IEEE Conference on Artificial Intellingence Applications, Miami, Florida, February 1991, pp. 134–140. The Institute of Electrical and Electronic Engineers (1991)
43. von Luck, K., Nebel, B., Peltason, C., Schmiedel, A.: BACK to consistency and incompleteness. In: Stoyan, H. (ed.) Proceedings of GWAI-1985—the 9th German Workshop on Artificial Intelligence, pp. 245–257. Springer, Heidelberg (1986)
44. Wright, J.R., Weixelbaum, E.S., Brown, K., Vesonder, G.T., Palmer, S.R., Berman, J.I., Moore, H.H.: A knowledge-based configurator that supports sales, engineering, and manufacturing at AT&T network systems. In: Proceedings of the Innovative Applications of Artificial Intelligence Conference, Washington, July 1993, pp. 183–193. American Association for Artificial Intelligence, Menlo Park (1993)

The KBMS Project and Beyond

Vinay K. Chaudhri[1], Igor Jurisica[2], Manolis Koubarakis[3], Dimitris Plexousakis[4], and Thodoros Topaloglou[5]

[1] SRI International, Menlo Park, CA, USA
[2] Ontario Cancer Institute, University Health Network, Toronto, Canada
[3] National and Kapodistrian University of Athens, Athens, Greece
[4] University of Crete and FORTH-ICS, Heraklion, Crete, Greece
[5] McGill University and Genome Quebec Innovation Center, Montreal, Canada

Abstract. The Knowledge Base Management Systems (KBMS) Project at the University of Toronto (1985-1995) was inspired by a need for advanced knowledge representation applications that require knowledge bases containing hundreds of thousands or even millions of knowledge units. The knowledge representation language Telos provided a framework for the project. The key results included conceptual modeling innovations in the use of semantic abstractions, representations of time and space, and implementation techniques for storage management, query processing, rule management, and concurrency control. In this paper, we review the key ideas introduced in the KBMS project, and connect them to some of the work since the conclusion of the project that is either closely related to or directly inspired by it.

1 Introduction

In the early nineties, the emergence of advanced applications such as Computer-Aided Design (CAD), software engineering, real-time control systems, and grand challenge projects such as the DARPA knowledge sharing project [1] and the Human Genome project [2] required technologies for creating and managing data representations with rich structure and ability for inference. The goal of the Knowledge Base Management Systems (KBMS) project, led by Prof. John Mylopoulos, was to develop the technology for construction and efficient access of large and shared knowledge bases (KBs). At that time, the state of the art in KB implementations was to use expert system shells or programming languages such as Lisp or Prolog. In the KBMS project, we investigated the use of database technology as a source for techniques for advancing the state of the art in constructing large knowledge base systems [3].

The knowledge representation language Telos provided the focal point of research in the KBMS project, and a framework for five Ph.D. [4,5,6,7,8] and three Masters [9,10] theses. We begin this paper with an overview of Telos and the key technical results of the project. We then discuss the current state of development of knowledge base systems most closely related to the problems investigated in the KBMS project.

2 The Knowledge Representation Language Telos

Since a detailed description of Telos is available in published papers [11], we will highlight here only its salient features.

Propositions: A proposition is the fundamental unit in a KB. A proposition is a triple, and can represent an individual or an attribute.

Structural Knowledge: The propositions in a KB are organized along three dimensions: aggregation, classification, and instantiation. The aggregation represents structured objects, the classification represents the class-subclass relationship, and the instantiation represents the class-instance relationship.

Classes can be instances of other classes, thus allowing meta-classes. Meta-classes are a mechanism for extending the representation model. Similarly, attributes can be instances of classes, called *attribute classes*, which provide a mechanism to impose constraints or specific semantics to attributes.

Temporal Knowledge: Every Telos proposition has an associated history time and a belief time. The history time of a proposition represents the lifetime of a proposition in the application domain (i.e., the lifetime of an entity or a relationship). A proposition's belief time, on the other hand, refers to the time when the proposition is believed by the KB, i.e., the interval between the moment the proposition is added to the KB and the time when its belief is terminated. In Telos, time is modeled using intervals that can be related using the relations in Allen's Interval Algebra [12]. The combination of constraint-based temporal representations and nontemporal ones was fully explored in the Ph.D. thesis of Koubarakis [4].

Assertional Knowledge: Telos provides an assertion language for the expression of deductive rules and integrity constraints. The assertion language is a first-order language with equality. Telos supports both static constraints (that apply to all states of a KB) and dynamic constraints (that apply to those temporal states that satisfy specific temporal predicates or to the transition between temporal states). Deductive rules are also explicitly associated with history and belief time intervals.

Telos supports primitive KB operations such as *Tell*, *Untell*, *Retell*, and *Ask* that can be used to query and update the KB. A possible-worlds semantics for Telos was defined in the Master's thesis of Plexousakis [10]. The semantics of Telos include an ontology of objects based on the property of existence, and proofs for the soundness, consistency, and completeness of a Telos KB.

Spatial Knowledge: The representation for the spatial knowledge in Telos was defined in the Ph.D. thesis of Topaloglou [7]. The spatial representation in Telos was accomplished through a library of meta-classes and meta-attributes that capture the semantics of spatial features of physical objects. The objects with a spatial extension can be *placed in* spaces, called *maps*, at variable scales and related to other spatial objects or constants through qualitative or quantitative relationships.

2.1 Telos Implementations

There have been four implementations of Telos. The first implementation was done as part of two Master's theses [9], carried out at the University of Toronto and ICS-FORTH, and covered all the features of Telos including reasoning with incomplete temporal information. The implementation language was Prolog with the temporal reasoning module implemented in C for efficiency. The temporal reasoning module implemented constraint satisfaction for a subset of Allen's Interval Algebra so that consistency checking remains polynomial. Nevertheless, the data complexity of query answering for the query language used is at least NP-hard [13].

The second implementation of Telos was done by an ICS-FORTH team led by Martin Doerr. The implementation was done in C++ and covered only the structural knowledge, and provided no support for assertional or temporal knowledge. This Telos prototype was used in the implementation of the Software Information Base [14].

The third implementation of Telos was based on the dialect O-Telos defined in the Ph.D. thesis of Manfred Jeusfeld at the University of Passau, and implemented in the ConceptBase system [15]. O-Telos omitted the history time component of Telos and its facilities for reasoning with incomplete information, and implemented only the structural and assertional knowledge features of Telos. Since 1995, ConceptBase has been continuously developed and it is freely available.[1] ConceptBase is currently applied at more than 500 sites worldwide for research as well as teaching and is the most complete implementation of Telos.

The fourth implementation, called Common Knowledge Base (CKB), was done by Bryan Kramer and Martin Stanley at the University of Toronto in the context of the APACS project. This implementation was done in C++ and a commercial object-oriented database management system, Versant, and covered only the structural knowledge [16].

2.2 Research on Implementation Techniques

The implementations of Telos mentioned above were done using the technology available at that time, did not support all the features of Telos, and, with the exception of ConceptBase, were not designed with scalability in mind. A significant effort in the KBMS project was devoted to addressing these limitations. Specifically, we investigated database techniques for implementing Telos, and subjected those techniques to rigorous performance evaluation [3]. We summarize here the techniques that were developed during the KBMS project.

Storage Management: The research goal of this task was to efficiently store on disk a KB that was too large to fit in the main memory. We developed a scheme called *Controlled Decomposition Model*, that could map the structural knowledge of a KB to a set of relations in a way that the information that was

[1] http://www-i5.informatik.rwth-aachen.de/CBdoc/

likely to be accessed together was stored together for efficient access, while the rest was split across multiple relations to support efficient storage and updates [17]. We extended the database techniques based on join indices to deal with the temporal dimension of a KB.

Query Processing: The research goal of this task was to develop semantic query optimization techniques for processing Telos queries [18]. We developed query simplification techniques for temporal knowledge, syntactic simplification techniques that apply knowledge of the class hierarchy, and techniques for generating query evaluation plans. We also developed a cost model for optimizing path queries in knowledge bases with structural and temporal knowledge. [19].

Concurrency Control: The goal of this task was to develop techniques to allow multi-user updates to a KB and was done as part of the Ph.D. thesis of Chaudhri [5]. The key result was a new locking protocol, called the *Dynamic Directed Graph policy*, that improved performance over the traditional two phase locking protocol by taking advantage of the assertional component of Telos.

Integrity Constraints: The research goal of this task was to develop an algorithm for efficient checking of integrity constraints for a Telos KB. This research was done as a part of the Ph.D. thesis of Plexousakis [6]. We developed techniques for compiling, simplifying and checking the violation of integrity constraints when there are updates to a KB. The technique took into account the temporal knowledge of Telos as well as potential interactions between rules and integrity constraints.

Case-based Reasoning: The research goal of this task was to develop a methodology for building large and complex case-based reasoning systems, and it was done as part of the Ph.D. thesis of Jurisica [8]. To achieve flexibility without reducing performance, we adapted an incremental view maintenance algorithm from database management systems [20] into a reasoning system called $\mathcal{TA}\mathbf{3}$, and successfully applied it to several biomedical domains [21,22,23,22,24,25]. The key components of $\mathcal{TA}\mathbf{3}$ included case representation in Telos, incremental query relaxation, modified k-nearest neighbor algorithm, and anytime retrieval algorithm.

3 Evolution of Knowledge Systems Since the KBMS Project

It is helpful to think of problems in the general area of constructing large knowledge systems in three broad classes: Content Modeling, Implemented Systems and Measurement and Evaluation. We now consider each of these subtopics in detail concentrating on research that took place after the KBMS project, and is either closely related to it or directly inspired by it. We notice that much of this work is at the core of research areas that have received much attention recently such as scientific data management or the Semantic Web.

3.1 Content Modeling

There are two aspects of content modeling: the knowledge representation language itself, and the KB content. In Telos, the temporal and spatial representations were considered part of the language itself. This is not the case in systems such as Cyc [26] in which the temporal knowledge and spatial knowledge are considered as subtheories in a KB - known as an upper ontology, and independent of the representation language. Here, we review the research on KR language features that were the subject of inquiry in Telos.

Structural Knowledge: In the mid-nineties, several KR research groups investigated languages such as CLASSIC [27], LOOM [28], and Ontolingua [29] that are in the tradition of KL-ONE [30]. In an effort to synthesize among the best features of various languages, Peter Karp at SRI International did a survey of frame-based knowledge representation languages prevalent at that time that led to the development of the Generic Frame Protocol [31] followed by the Open Knowledge Base Connectivity Model [32].

The World-Wide Web (WWW) gave rise to a whole new group of languages that are based on semantic abstractions and were designed as a foundation for the Semantic Web [33]. Building on the WWW technologies (for example, Uniform Resource Indicators) and the past research on semantic networks and frame-based knowledge representation languages, Resource Description Framework (RDF) was created [34] and became a standard of the World Wide Web Consortium (W3C) in 2004.[2] With a similar emphasis, the DARPA Agent Markup Language program[3] (DAML) focused on standardizing expressive knowledge representation languages for the WWW. At the same time, another group of researchers, mainly based in Europe, developed a competitor ontology language called the Ontology Interchange Language (OIL) [35]. The interaction among DAML and OIL led to the language DAML+OIL, which eventually became the Ontology Web Language or OWL that became a W3C recommendation in 2004.[4]

Looking back at the gradual development of the languages mentioned above, it is easy to discern the influence of Telos and the KL-ONE tradition in RDF and the languages building on them. RDF triples bear a close resemblance to Telos propositions. The abstraction mechanisms adopted in these languages had already been incorporated as modeling primitives in the representation framework of Telos since the late 1980s.

Assertional Knowledge: Early efforts to combine an assertion language with structural knowledge predate Telos and the KBMS project (see, e.g., KRYPTON [36]); such languages are now part of several implemented systems [29,26]. In parallel to work on Telos, there have also been various other proposals of languages combining an assertional and structural knowledge, many of them

[2] `http://www.w3.org/RDF/`
[3] `http://www.daml.org`
[4] `http://www.w3.org/2004/OWL/`

coming under the label "deductive and object-oriented" languages. Perhaps the most theoretically elegant language in this family is F-Logic [37]. F-Logic, like all other deductive and object-oriented languages, originated from the database and logic programming traditions, and accounts in a clean and declarative fashion for most of the structural aspects of object-oriented and frame-based languages.

F-Logic, however, does not allow for temporal or spatial knowledge in its assertion language in the way that Telos supports. An additional distinguishing feature of the assertional component of Telos, not present in related languages, is the coupling of the integrity constraints and deductive rules with the structural knowledge. In Telos, rules and constraints are treated uniformly and can be attached to classes at any level of the class hierarchy.

More recently, there have been several proposals for rule languages for the Semantic Web. For example, Semantic Web Rule Language (SWRL) deals with issues such as rule definition based on OWL.[5] Semantic Inferencing for Large Knowledge or SILK is an effort to define an expressive rule language that addresses issues such as reasoning with processes and defaults.[6]. The Rule Markup Language (RuleML) initiative aims at defining a common rule interchange and markup language based on XML and RDF.[7] The problems studied in the context of the specification, semantics and use of the assertion language of Telos are thus recast and expanded in the context of reasoning on the Semantic Web.

Temporal Knowledge: The rich temporal knowledge representation concepts of Telos (history time, belief time, interval-based incompleteness of historical knowledge) did not find any followers in the deductive and object-oriented (or frame-based) families of languages. There has been interesting related work in other areas.

The distinction between history and belief time had already been made explicitly in the area of temporal relational databases [38,39] before Telos was put forward (the corresponding terms used were *valid time* and *transaction time*) and Telos had benefited from this work. The area of temporal databases saw an explosion of activity culminating in the specification of the query language TSQL2 [40], a temporal extension of the SQL-92 standard. In the deductive database and logic programming community, the most comprehensive proposal to introduce valid and transaction time using event calculus is by Sripada [41].

Gutierrez and colleagues [42,43,44] have proposed to extend RDF triples of the form (s, p, o) with an additional temporal label t that is a natural number. The resulting quad (s, p, o, t) is called a *temporal RDF triple* and denotes the fact that the triple (s, p, o) is valid at time t. Based on this definition, [42,43,44] define a *temporal RDF graph* as a set of temporal RDF triples, and study problems of semantics and computational complexity of query answering. This representation directly maps to the representation of propositions in Telos.

The temporal description logics introduce a *concrete domain* to model time together with appropriate definitions for concepts, roles and features [45]. Work

5 `http://www.w3.org/Submission/SWRL/`

6 `http://silk.projects.semwebcentral.org/`

7 `http://www.ruleml.org/`

here has concentrated mostly on issues of semantics and reasoning in these logics with little emphasis on implementations.

Starting with Cyc,[8] there has also been work on ontologies of time [9]. The time ontology in Cyc is one of the most comprehensive representations of time available today and has been reused outside the context for which it was originally created [46].

Spatial Knowledge: Numerous spatial representations and calculi to support efficient reasoning with spatial knowledge are now available that were not available at the time of the KBMS project [47]. The most comprehensive representation of space in an implemented system is in the Cyc KB. The Cyc KB provides a first-order axiomatization of basic spatial representational primitives. It includes axiomatization of more than 65 spatial predicates, which include 15 different kinds of containment and 7 different kinds of covering. It also supports representations for shapes, boundaries, regions, and convex hulls [48].

The Semantic Web language RDF has recently been extended to represent spatial knowledge. In the system SPAUK [49], geometric attributes of a resource (e.g., location of a gas station) are represented in RDF by introducing a blank node for the geometry, specifying the geometry using the Geography Markup Language,[10] and associating the blank node with the resource using Geography Encoded Objects for RSS feeds.[11] Queries in SPAUK are expressed in the SPARQL query language utilizing geometric vocabularies and ontologies [47]. In a similar spirit, Perry defines an extension of SPARQL, called SPARQL-ST, that allows one to query spatial and nonspatial data with a time dimension [50].

3.2 Implemented Systems

We review here the implemented systems to which the graduates of the KBMS project have contributed. These systems are closely related to the spirit of the KBMS project.

The PERK System: The PERK (or Persistent Knowledge) system at SRI International was implemented to enable collaborative construction of KBs [51]. It addressed the issues of storage management, concurrency control, graphical editing, and application programming interfaces. The PERK system supports only structural knowledge, and has no support for assertional, temporal or spatial knowledge. We consider here the topics of storage management and concurrency control since they are most closely related to the implementation techniques of Telos KBMS.

The storage system in PERK is aimed at allowing incremental loading and saving of a large KB into the main memory. It achieves this by submerging

[8] `http://www.opencyc.org/`
[9] `http://www.w3.org/TR/2006/WD-owl-time-20060927/`
[10] `http://www.opengeospatial.org/standards/gml`
[11] `http://georss.org/`

the commercial Oracle database management system in an existing Lisp-based frame-based representation system. Due to impedance mismatch between the in-memory representation of Lisp objects and a relational database, and to facilitate evolution of the schema, PERK represents the content of a frame using a compressed string representation. Since a string representation is not in the first normal form, it is unable to support query processing directly by the database. All the query processing is then done in memory using the frame system. To facilitate fast retrieval, PERK adds indices for frequently accessed slots. The spirit of this design is very close to the controlled decomposition model used in the Telos KBMS.

The concurrency control scheme in PERK is based on an optimistic concurrency control approach: the users are allowed to make divergent updates to the KB; and once they are satisfied with their changes, they attempt to commit their updates; any conflicting updates are detected and they must resolve the conflicts. This model seemed preferable to locking-based approach advocated in the Telos KBMS because it allowed greater collaboration among the contributors. The conflict detection is done based on a change log that is organized at the granularity of knowledge editing operations and not low-level storage operations in a database. The PERK system is in extensive use as the back end of SRI's EcoCyc system [52].

The OPM Tools Suite: The Object Protocol Model (OPM) project [53] at the Lawrence Berkeley National Laboratory utilizes a semantic data model to describe the semantics of the entities in biological databases and build advanced query mechanisms for biological data. The OPM project shares significant similarities with the Telos knowledge management project.

OPM is an object-based data model that is very similar to Telos and is used to describe a database at a conceptual level in terms of classes, attributes, and relationships. By representing the semantics of the data, it enables scientific users to create, query, and integrate databases without being bogged down by implementation and system-level details. The gap between the user view and the implementation view of a database is bridged by data management tools that are driven by rich meta-data represented in OPM. The OPM suite includes tools for creating and querying databases, adding a query interface to an existing database, querying multiple databases, and extending databases with application-specific data types and methods. Forward development of an OPM database results in the creation of a relational database and object-relational (O-R) mapping meta-data that are used to reformulate object queries to relational ones, and reconstruct objects from the results of a relational query plan. The O-R mapping of OPM shares striking similarities to the controlled decomposition model, although they have been developed independently.

The mechanism for model extensibility in OPM, needed to support methods and application-specific types, such as DNA sequences and images, was realized following an approach similar to the Telos approach, i.e., adding a new meta-class at the language level, and complemented by a robust integration framework and implementation infrastructure [54].

The RDF Suite: Due to the expansion of the WWW, there is a significant need to enable querying over heterogeneous information sources. One effort to meet this need is the ICS-FORTH RDFSuite [55], which provides services for loading, parsing, schema-aware storage, querying, viewing, and updating of RDF/S resource descriptions and schemas. The design of RDFSuite was influenced by the design decisions of the Telos KBMS, in particular with respect to storage management and querying. RDFSuite comprises (a) the Validating RDF Parser: a parser supporting semantic validation of both RDF/S resource descriptions and schemas, (b) the RDF Schema Specific DataBase: a store exploiting a variety of Object-Relational (SQL3) representations to manage RDF/S resource descriptions and schemas, (c) the RDF Query Language: a declarative language for uniformly querying RDF/S resource descriptions and schemata, (d) the RDF View Language: the first declarative language for creating virtual RDF/S resource descriptions and schemas, and (e) the RQL Graphical Query Generator: a user interface generating minimal declarative queries by taking into account the browsing actions in an RDF/S schema during a user navigation session.

The $\mathcal{TA}$3 System: The $\mathcal{TA}$**3** system was developed as part of the Ph.D. thesis of Jurisica [8] and has undergone significant evolution. The $\mathcal{TA}$**3** system has been expanded by using incremental query relaxation and anytime retrieval algorithm [20], reimplementing the system in Java, and using the IBM DB2 relational database for persistent storage [23]. Further expansion covered support to handle images [22,56,24,57], improved performance with data mining [25,58], and improved accuracy with an ensemble of classifiers [59].

The $\mathcal{TA}$**3** system currently handles the scale and the size of data for which the KBMS project was envisioned. As a specific example, it is being used to store and analyze protein crystallization experiments [60,23,58]. In its current use, there are 12,000 protein crystallization experiments, each of which has 9,216 attributes, and the data is derived from 110,592,000 images [61,62]. The repository grows at a rate of more than 200 experiments each month. Before $\mathcal{TA}$**3** can suggest crystallization optimization strategies for a novel protein, we have to compute 12,375 image features and automatically classify images into 10 possible categories [56,63,61,62].

The key to scaling the techniques that were developed in the KBMS project for this new application has been the use of a scalable computational infrastructure. Although the $\mathcal{TA}$**3** system can run on a regular Unix or Windows server, the image processing runs on the World Community Grid[12] and the post-processing is done on a 1,344-core Linux cluster.

Another large-scale application of the $\mathcal{TA}$**3** system involves estimating a job runtime [64]. The application domain considered scheduling Functional Regression Tests (FRT) for the IBM DB2 Universal Database product Version 8.2 (DB2 UDB) [65]. A job runtime can be affected by a large number of both job and machine characteristics. Scalable performance is critical, as there are more than 50,000 jobs to test for each version of DB2 UDB in a grid, which comprises about

[12] http://www.worldcommunitygrid.org

300 machines with different configurations. A case is represented as a record that includes job information, machine information, and runtime information. Case retrieval and adaptation uses the priorities of job and both static and dynamic machine characteristics. Our experimental results show that for more than 90% of jobs, the estimation error is 45% or less, and the average estimation error is at most 22%. Applying the system to FRT, we achieved average performance improvements of 20% to 50%. In the worst case, we still achieved a 13% to 36% performance improvement [65].

3.3 Measurement and Evaluation

Performance evaluation of implementation techniques was a central methodology in the KBMS project. We discuss here innovations in measurement and evaluation of knowledge-based systems.

Metrics on KB Size: In the world of databases, the size of the data measured in records or bytes is a very good indicator of the scale of the problem. But, a similar measure such as “millions of knowledge units” is not always meaningful in the context of KBs. For example, a KB containing millions of simple facts may have less knowledge content than a KB with a small number of axioms. As an approximate measure, one can use the number of axioms in a KB as one measure of competence. competence.

One possible approach to measuring the size was investigated in DARPA’s High Performance KBs (HPKB) project in which the measure of axiom counts was refined to include axiom categories [66]:

1. *Constants* are any names in the KB, whether an individual, class, relation, function, or a KB module.
2. *Structural statements* are ground statements about the semantic abstractions in a KB, for example, subclass-of, instance-of, domain, and range assertions.
3. *Ground facts* are any statement without a variable.
4. *Implications* include any nonground statement that has an *implies* (a ground statement that contains an *implication* is counted as a ground statement).
5. *Non ground, non implications* are statements that contain variables but not an implication.

While axiom categories are an improvement over measuring the size in terms of *knowledge units*, they are still imperfect; a larger number of axioms in a category does not alway imply a greater amount of knowledge. As a methodological improvement, Vulcan’s Project Halo measures the size of a KB in terms of the questions it is able to answer on a standardized college-level test [67]. While the approach of using a standardized test is a significant improvement, we believe new ways of measuring the KB content and quality are needed that are applicable to more general classes of systems.

Knowledge Reuse Metrics: A central claim in building the content for large KBs is that as we add more content, things get easier, as the content added later builds on what already exists. This is a claim that makes intuitive sense because when new knowledge is to be added to a KB, relevant terms may already exist and some knowledge may be available through inheritance. One innovative way to test this claim is to study how knowledge is reused in a large KB [68] that was tried in DARPA's HPKB project.

The knowledge reuse metric can be defined as follows. Suppose one wishes to add a new piece of knowledge to a KB. Every item i one wishes to add to the KB contains $n(i)$ terms, $k(i)$ of which are already in the KB, and support is $s(i) = k(i)/n(i)$. Adding new terms to a KB changes the size of the KB, and the support offered by the KB for future axioms might be higher because new terms were added. Thus, support is indexed by versions of the KB: $s(i, j) = k(i, j)/n(i)$ is the support provided by version j of the KB for concept i.

Although the idea of knowledge sharing has been in the literature for many years [1], the reuse metric was one of the first attempts to empirically study the claim. The results in using this metric suggested that the answer depends on the kind of prior knowledge, who is using it, and what it is used for. There is still lot of room for further understanding of the KB construction process: How long will a knowledge engineer hunt for a relevant term or axiom in a prior ontology? How rapidly do KBs diverge from available ontologies if knowledge engineers do not find the terms they need in the ontologies? By what process does a knowledge engineer reuse not an individual term but a larger fragment of an ontology, including axioms? How does a very general ontology inform the design of KBs, and what factors affect whether knowledge engineers take advantage of the ontology? Why do prior ontologies apparently provide less support for encoding axioms than for encoding test questions?

Benchmarks for Knowledge Base Systems: At the time of the KBMS project, hardly any data sets were available for testing tools and algorithms that we were investigating. In recent years, some progress has been made on this front, and we review two such efforts.

The Lehigh University Benchmark (LUBM) is a benchmark for large OWL KBs [69]. The LUBM features an ontology for the university domain, synthetic OWL data scalable to an arbitrary size, fourteen extensional queries representing a variety of properties, and several performance metrics. The LUBM can be used to evaluate systems with different reasoning capabilities and storage mechanisms. It has been used for memory-based systems as well as systems with persistent storage.

The OpenRuleBench[13] is a suite of benchmarks for analyzing the performance and scalability of different rule engines. It has been tested on five different technologies: Prolog, deductive databases, production rules, triple engines, and general KBs. It examines how different systems scale for a number of common problem sets that involve reasoning such as recursion and negation. It also

[13] http://rulebench.projects.semwebcentral.org/

examines, by interpolation, how these technologies or their successors might perform on the WWW scale.

There is no existing benchmark to characterize the performance of temporal and spatial reasoning systems, but there is recent interest in the research community to define such a benchmark [70].

We believe that further work on benchmarking and evaluating the implementations of KB systems is essential to continued science and engineering of KBMS implementations.

4 Summary and Conclusions

The Telos KBMS project attempted an extensive experimentation of applying database management techniques to implementation of KBs. The project fulfilled its goals by at least two measures. First, it generated five Ph.D. theses, and three Master's theses, and trained highly qualified personnel who later contributed to a number of visible projects. Second, it emphasized key themes in knowledge base management that have sustained the test of time. The key findings of the project were the following:

1. A rich representation language that combines semantic abstractions of classification, instantiation, aggregation with deductive rules, integrity constraints with temporal and spatial representation defined as part of the language is essential for constructing large knowledge bases.
2. To achieve efficient implementation of large knowledge bases, we will need to rely on implementation techniques from database management systems such as storage management, query processing, concurrency control, rule management, and view maintenence.
3. Performance evaluation is core methodology for testing the implementations for large KB systems.

The semantic abstractions considered in the Telos knowledge representation language are at the core of most modern representation systems. There are languages and systems that specialize in various advanced features such as object-oriented and deductive representations (for example, F-Logic), rule and constraint management (for example, relational and deductive databases) or temporal reasoning (for example, Allen's calculus), but there is no unified representation and reasoning theory that combines all the features that were considered in Telos into one language.

The adoption of systems such as PERK, OPM, and $\mathcal{TA}\mathbf{3}$ suggests that the bio-medical research community has been the earliest adopter of expressive representation languages and the database techniques of the sort investigated in the Telos KBMS project. The recent work on the Semantic Web is leveraging the same ideas and technology - the theory and practice behind the RDF framework (e.g., as realized in the RDFSuite) is just one such instance.

Performance evaluation of knowledge base systems has now become a central methodology, especially in government funded projects in the United States as exemplified by the HPKB project. The research community has started to define benchmarks for characterizing the performance for isolated systems features (for example, LUBM and OpenRuleBench), and there is growing need and acceptance for benchmarking and performance evaluation of knowledge base systems.

Reflecting on our work on the KBMS project, we can say that due to the WWW and the proliferation of scientific data, the knowledge management challenges today are even more real and significant than twenty years ago. As a result, the concepts, techniques, and methods investigated in the KBMS project will continue to have very high relevance for many years to come.

References

[1] Neches, R., Fikes, R., Finin, T.W., Gruber, T.R., Patil, R.S., Senator, T.E., Swartout, W.R.: Enabling technology for knowledge sharing. AI Magazine 12(3), 36–56 (1991)

[2] Frenkel, K.A.: The Human Genome Project and informatics. Communications of the ACM 34(11), 41–51 (1991)

[3] Mylopoulos, J., Chaudhri, V.K., Plexousakis, D., Shrufi, A., Topaloglou, T.: Building knowledge base management systems. The VLDB Journal 5(4), 238–263 (1996)

[4] Koubarakis, M.: Foundations of Temporal Constraint Databases. PhD thesis, Computer Science Division, Dept. of Electrical and Computer Engineering, National Technical University of Athens (February 1994)

[5] Chaudhri, V.K.: Transaction Synchronization in Knowledge Bases: Concepts, Realization and Quantitative Evaluation. PhD thesis, University of Toronto, Toronto (January 1995)

[6] Plexousakis, D.: Integrity Constraint and Rule Maintenence in Temporal Deductive Knowledge Bases. PhD thesis, University of Toronto, Toronto (1996)

[7] Topalogou, T.: On the Representation of Spatial Knowledge in Knowledge Bases. PhD thesis, University of Toronto, Toronto (1996)

[8] Jurisica, I.: TA3: Theory, Implementation, and Applications of Similarity-Based Retrieval for Case-Based Reasoning. PhD thesis, University of Toronto, Department of Computer Science, Toronto, Ontario (1998)

[9] Topaloglou, T., Koubarakis, M.: Implementation of Telos: Problems and Solutions. Technical Report KRR-TR-89-8, Dept. of Computer Science, University of Toronto (1989)

[10] Plexousakis, D.: An Ontology and a Possible-Worlds Semantics for Telos. Master's thesis, Dept. of Computer Science, University of Toronto (1990)

[11] Mylopoulos, J., Borgida, A., Jarke, M., Koubarakis, M.: Telos: A language for representing knowledge about information systems. ACM Transactions on Information Systems 8(4), 325–362 (1990)

[12] Allen, J.: Maintaining knowledge about temporal intervals. Communications of the ACM 26(11), 832–843 (1983)

[13] Koubarakis, M.: The complexity of query evaluation in indefinite temporal constraint databases. Theoretical Computer Science 171, 25–60 (1997); Special Issue on Uncertainty in Databases and Deductive Systems, Lakshmanan, L.V.S. (ed.)

[14] Constantopoulos, P., Doerr, M., Vassiliou, Y.: Repositories for software reuse: The software information base. In: Information System Development Process, pp. 285–307 (1993)

[15] Jarke, M., Gallersdörfer, R., Jeusfeld, M.A., Staudt, M.: ConceptBase - A deductive object base for meta data management. Journal of Intelligent Information Systems 4(2), 167–192 (1995)

[16] Wang, H., Mylopoulos, J., Kusniruk, A., Kramer, B., Stanley, M.: KNOWBEL: New tools for expert system development. In: Bourbakis, N.G. (ed.) Developement of Knowledge-Based Shells. Advanced Series on Artificial Intelligence, World Scientific, Singapore (1993)

[17] Topaloglou, T.: Storage management for knowledge bases. In: CIKM 1993: Proceedings of the Second International Conference on Information and Knowledge Management, pp. 95–104. ACM, New York (1993)

[18] Topaloglou, T., Illarramendi, A., Sbattella, L.: Query optimization for KBMSs: Temporal, syntactic and semantic transformantions. In: Golshani, F. (ed.) Proceedings of the Eighth International Conference on Data Engineering, Tempe, Arizona, February 3-7, 1992, pp. 310–319. IEEE Computer Society, Los Alamitos (1992)

[19] Shrufi, A., Topaloglou, T.: Query processing for knowledge bases using join indices. In: Proceedings of the 4th International Conference on Information and Knowledge Management, Baltimore (November 1995)

[20] Jurisica, I., Glasgow, J., Mylopoulos, J.: Incremental iterative retrieval and browsing for efficient conversational CBR systems. International Journal of Applied Intelligence 12(3), 251–268 (2000)

[21] Jurisica, I., Mylopoulos, J., Glasgow, J., Shapiro, H., Casper, R.F.: Case-based reasoning in IVF: Prediction and knowledge mining. Artif. Intell. Med. 12(1), 1–24 (1998)

[22] Jurisica, I., Rogers, P., Glasgow, J., Collins, R., Wolfley, J., Luft, J., DeTitta, G.: Improving objectivity and scalability in protein crystallization: Integrating image analysis with knowledge discovery. IEEE Intelligent Systems Journal, Special Issue on Intelligent Systems in Biology, 26–34 (November/December 2001)

[23] Jurisica, I., Rogers, P., Glasgow, J., Fortier, S., Luft, J., Wolfley, J., Bianca, M., Weeks, D., DeTitta, G.T.: Intelligent decision support for protein crystal growth. IBM Systems Journal, Special Issue on Deep Computing for Life Sciences 40(2), 394–409 (2001)

[24] Jurisica, I., Glasgow, J.: Application of case-based reasoning in molecular biology. Artificial Intelligence Magazine, Special issue on Bioinformatics 25(1), 85–95 (2004)

[25] Arshadi, N., Jurisica, I.: Integrating case-based reasoning systems with data mining techniques for discovering and using disease biomarkers. IEEE Transactions on Knowledge and Data Engineering. Special Issue on Mining Biological Data 17(8), 1127–1137 (2005)

[26] Lenat, D.B., Guha, R.V., Pittman, K., Pratt, D., Shepherd, M.: Cyc: Toward programs with common sense. Commun. ACM 33(8), 30–49 (1990)

[27] Borgida, A., Brachman, R., McGuiness, D., Resnick, L.: CLASSIC: A structural data model for objects. In: Proceedings of ACM SIGMOD International Conference on Management of Data, pp. 58–67 (1989)

[28] MacGregor, R.M., Brill, D.: Recognition algorithms for the LOOM classifier. In: Proceedings of the National Conference on Artificial Intelligence (AAAI), pp. 774–779 (1992)

[29] Farquhar, A., Fikes, R., Rice, J.: The ontolingua server: a tool for collaborative ontology construction. Int. J. Hum.-Comput. Stud. 46(6), 707–727 (1997)

[30] Brachman, R., Schmolze, J.: An overview of the KL-ONE knowledge representation system. Cognitive Science 9(2), 171–216 (1985)

[31] Karp, P.D., Myers, K.L., Gruber, T.R.: The generic frame protocol. In: IJCAI, vol. 1, pp. 768–774 (1995)

[32] Chaudhri, V.K., Farquhar, A., Fikes, R., Karp, P.D., Rice, J.: OKBC: A programmatic foundation for knowledge base interoperability. In: AAAI/IAAI, pp. 600–607 (1998)

[33] Berners-Lee, T., Hendler, J., Lassila, O.: The semantic web. Scientific American 284(5), 34–43 (2001)

[34] Lassila, O.: The resource description framework. IEEE Intelligent Systems 15(6), 67–69 (2000)

[35] Fensel, D., van Harmelen, F., Horrocks, I., McGuinness, D.L., Patel-Schneider, P.F.: OIL: an ontology infrastructure for the semantic web. IEEE Intelligent Systems 16(2), 38–45 (2001)

[36] Brachman, R.J., Levesque, H.J., Fikes, R.: Krypton: Integrating terminology and assertion. In: AAAI, pp. 31–35 (1983)

[37] Kifer, M., Lausen, G.: F-Logic: A higher-order language for reasoning about objects, inheritance, and scheme. In: Proceedings of ACM SIGMOD International Conference on Management of Data, pp. 134–146 (1989)

[38] Snodgrass, R., Ahn, I.: A taxonomy of time in databases. In: Proceedings of ACM SIGMOD International Conference on Management of Data, pp. 236–246 (1985)

[39] Snodgrass, R.: The temporal query language TQuel. ACM Transcactions on Database Systems 12(2), 247–298 (1987)

[40] Snodgrass, R.T. (ed.): The TSQL2 Temporal Query Language. Kluwer, Dordrecht (1995)

[41] Sripada, S.M.: A logical framework for temporal deductive databases. In: Bancilhon, F., DeWitt, D.J. (eds.) Fourteenth International Conference on Very Large Data Bases, Proceedings, Los Angeles, California, USA, August 29-September 1, 1988, pp. 171–182. Morgan Kaufmann, San Francisco (1988)

[42] Gutierrez, C., Hurtado, C., Vaisman, A.: Introducing time into RDF. IEEE Transactions on Knowledge and Data Engineering 19(2), 207–218 (2007)

[43] Gutierrez, C., Hurtado, C., Vaisman, R.: Temporal RDF. In: European Conference on the Semantic Web, pp. 93–107 (2005)

[44] Hurtado, C., Vaisman, A.: Reasoning with temporal constraints in RDF. In: Principles and Practice of Semantic Web Reasoning, pp. 164–178. Springer, Heidelberg (2006)

[45] Lutz, C., Milicic, M.: A tableau algorithm for description logics with concrete domains and general tboxes. J. Autom. Reasoning 38(1-3), 227–259 (2007)

[46] Chaudhri, V.K., Stickel, M.E., Thomere, J.F., Waldinger, R.J.: Reusing prior knowledge: Problems and solutions. In: Proceedings of the AAAI Conference on Artificial Intelligence (2000)

[47] Cohn, A.G., Bennett, B., Gooday, J.M., Gotts, N.: RCC: A calculus for region based qualitative spatial reasoning. GeoInformatica, 275–316 (1997)

[48] Uribe, T.E., Chaudhri, V.K., Hayes, P.J., Stickel, M.E.: Qualitative spatial reasoning for question-answering: Axiom reuse and algebraic methods. In: Proceedings of the AAAI Spring Symposium on Mining Answers from Texts and Knowledge Bases (2002)

[49] Kolas, D., Self, T.: Spatially-augmented knowledge base. In: Aberer, K., Choi, K.-S., Noy, N., Allemang, D., Lee, K.-I., Nixon, L., Golbeck, J., Mika, P., Maynard, D., Mizoguchi, R., Schreiber, G., Cudré-Mauroux, P. (eds.) ASWC 2007 and ISWC 2007. LNCS, vol. 4825, pp. 785–794. Springer, Heidelberg (2007)

[50] Perry, M.: A Framework to Support Spatial, Temporal and Thematic Analytics over Semantic Web Data. PhD thesis, Wright State University (2008)

[51] Karp, P.D., Chaudhri, V.K., Paley, S.M.: A collaborative environment for authoring large knowledge bases. J. Intell. Inf. Syst. 13(3), 155–194 (1999)

[52] Karp, P.D., Riley, M., Saier, M., Paulsen, I.T., Collado-Vides, J., Paley, S., Pellegrini-Toole, A., Bonavides, C., Gama-Castro, S.: The EcoCyc database. Nucleic Acids Research 30(1), 56–58 (2002)

[53] Chen, I.M.A., Kosky, A., Markowitz, V.M., Szeto, E., Topaloglou, T.: Advanced query mechanisms for biological databases. In: ISMB 1998: Proceedings of the 6th International Conference on Intelligent Systems for Molecular Biology, pp. 43–51. AAAI Press, Menlo Park (1998)

[54] Topaloglou, T., Kosky, A., Markowitz, V.M.: Seamless integration of biological applications within a database framework. In: Proceedings of the Seventh International Conference on Intelligent Systems for Molecular Biology, pp. 272–281. AAAI Press, Menlo Park (1999)

[55] Alexaki, S., Christophides, V., Karvounarakis, G., Plexousakis, D., Tolle, K.: The ICS-FORTH RDF Suite: Managing voluminous RDF description bases. In: Proceedings of the 2nd International Workshop on the Semantic Web (2001)

[56] Cumbaa, C.A., Lauricella, A., Fehrman, N., Veatch, C., Collins, R., Luft, J., DeTitta, G., Jurisica, I.: Automatic classification of sub-microlitre protein-crystallization trials in 1536-well plates. Acta Crystallogr. D Biol. Crystallogr. 59(Pt 9), 1619–1627 (2003); 22805983 0907-4449 Journal Article

[57] Acton, B.M., Jurisicova, A., Jurisica, I., Casper, R.F.: Alterations in mitochondrial membrane potential during preimplantation stages of mouse and human embryo development. Mol. Hum. Reprod. 10(1), 23–32 (2004)

[58] Jurisica, I., Wigle, D.A.: Knowledge Discovery in Proteomics. Mathematical Biology and Medicine. Chapman and Hall/CRC Press (2006)

[59] Arshadi, N., Jurisica, I.: An ensemble of case-based classifiers for high-dimensional biological domains. In: Muñoz-Ávila, H., Ricci, F. (eds.) ICCBR 2005. LNCS, vol. 3620, pp. 21–34. Springer, Heidelberg (2005)

[60] Jurisica, I., Rogers, P., Glasgow, J., Fortier, S., Collins, R., Wolfley, J., Luft, J., DeTitta, G.T.: High throughput macromolecular crystallization: An application of case-based reasoning and data mining. In: Johnson, L., Turk, D. (eds.) Methods in Macromolecular Crystallography, Kluwer Academic Publishers, Dordrecht (2000)

[61] Snell, E., Lauricella, A., Potter, S., Luft, J., Gulde, S., Collins, R., Franks, G., Malkowski, M., Cumbaa, C., Jurisica, I., DeTitta, G.T.: Establishing a training set through the visual analysis of crystallization trials Part II: Crystal examples. Acta Crystallographica D (2008)

[62] Snell, E., Luft, J., Potter, S., Lauricella, A., Gulde, S., Malkowski, M., Koszelak-Rosenblum, M., Said, M., Smith, J., Veatch, C., Collins, R., Franks, G., Thayer, M., Cumbaa, C., Jurisica, I., DeTitta, G.T.: Establishing a training set through the visual analysis of crystallization trials Part I: 150,000 images. Acta Crystallographica D (2008)

[63] Cumbaa, C., Jurisica, I.: Automatic classification and pattern discovery in high-throughput protein crystallization trials. J. Struct. Funct. Genomics 6(2-3), 195–202 (2005)

[64] Xia, E., Jurisica, I., Waterhouse, J., Sloan, V.: Runtime estimation using the case-based reasoning approach for scheduling in a grid environment. J. ACM (submitted)

[65] Xia, E., Jurisica, I., Waterhouse, J., Sloan, V.: Runtime estimation using a case-based reasoning system for scheduling in a grid environment. IBM Invention Disclosure (2007)

[66] Pease, A., Chaudhri, V.K., Lehman, F., Farquhar, A.: Practical Knowlege Representation and the DARPA High Performance Knowledge Base Project. In: Seventh International Conference on Principles of Knowledge Representation and Reasoning, Breckenridge, CO (2000)

[67] Friedland, N., Allen, P., Mathews, G., Whitbrock, M., Baxter, D., Curts, J., Shepard, B., Miraglia, P., Angele, J., Staab, S., Moench, E., Opperman, H., Wenke, D., Israel, D., Chaudhri, V., Porter, B., Barker, K., Fan, J., Chaw, S.Y., Yeh, P., Tecuci, D., Clark, P.: Project Halo: Towards a Digital Aristotle. The AI Magazine (2004)

[68] Cohen, P., Chaudhri, V.K., Pease, A., Schrag, B.: Does prior knowledge facilitate the development of knowledge-based systems. In: Proceedings of the AAAI 1999, pp. 221–226 (1999)

[69] Guo, Y., Pan, Z., Heflin, J.: LUBM: A benchmark for OWL knowledge base systems. The Journal of Web Semantics 3(2), 158–182 (2005)

[70] Nebel, B.: Benchmarking of qualitative temporal and spatial reasoning systems. In: AAAI Spring Symposium, AAAI Press, Menlo Park (2009)

Using the ConGolog and CASL Formal Agent Specification Languages for the Analysis, Verification, and Simulation of i* Models

Alexei Lapouchnian[1] and Yves Lespérance[2]

[1] Department of Computer Science, University of Toronto, Toronto, ON M5S 3G4, Canada
alexei@cs.toronto.edu
[2] Department of Computer Science and Engineering, York University, Toronto, ON M3J 1P3, Canada
lesperan@cse.yorku.ca

Abstract. This chapter describes an agent-oriented requirements engineering approach that combines informal *i** models with formal specifications in the multiagent system specification formalisms ConGolog and its extension CASL. This allows the requirements engineer to exploit the complementary features of the frameworks. *i** can be used to model social dependencies between agents and how process design choices affect the agents' goals. ConGolog or CASL can be used to model complex processes formally. We introduce an intermediate notation to support the mapping between *i** models and ConGolog/CASL specifications. In the combined *i**-CASL framework, agents' goals and knowledge are represented as their subjective mental states, which allows for the formal analysis and verification of, among other things, complex agent interactions and incomplete knowledge. Our models can also serve as high-level specifications for multiagent systems.

This volume is dedicated to John Mylopoulos. Yves was fortunate to have John as his Master's thesis supervisor 30 years ago and John is Alexei's current Ph.D. thesis supervisor. The work described in this paper fits perfectly in the model-based approach to software/systems engineering that John developed and promoted throughout his career. His vision, with its roots in knowledge representation research, its embrace of ideas from social science, and its insights into the "model-based" future of software/systems engineering continues to inspire us. Thanks John, for all the inspiration and mentoring.

1 Introduction

*i** [29] is an informal diagram-based language for early-phase requirements engineering that supports the modeling of social and intentional dependencies between agents and how process design choices affect the agents' goals, both functional and non-functional. It has become clear that such social and organizational issues play an important role in many domains and applications. However, *i** is not a formal language, has inadequate precision, and thus provides limited support for describing and analyzing complex processes. While it is possible to informally analyze small systems, formal analysis is needed for realistically-sized ones.

To alleviate this, we first propose an approach that integrates *i** with a formal multiagent system specification language, *ConGolog* [5, 13], in the context of agent-oriented requirements engineering. ConGolog is an expressive formal language for process specification and agent programming. It supports the formal specification of complex multiagent systems, but lacks features for modeling the rationale behind design choices available in *i**. In this paper, we show how *i** and ConGolog can be used in combination. The *i** framework will be used to model different alternatives for the desired system, to analyze and decompose the functions of the different actors, and to model the dependency relationships between the actors and the rationale behind process design decisions. The ConGolog framework will be used to formally specify the system behaviour described informally in the *i** model. The ConGolog model will provide more detailed information about the actors, tasks, processes, and goals in the system, and the relationships between them. Complete ConGolog models are executable and this will be used to validate the specifications by simulation. To bridge the gap between *i** and ConGolog models, an *intermediate notation* involving the use of process specification annotations in *i** SR diagrams will be introduced [26, 27]. We will describe how such *annotated SR (ASR) diagrams* can be systematically mapped into ConGolog formal specifications that capture their informal meaning, and support validation through simulation and verification. The annotations are not used to capture design-level information, but to obtain a more complete and precise model of the domain.

Its support for modeling intentional notions such as goals makes the *i** notation especially suited for developing multiagent systems, e.g., as in the Tropos agent-oriented development framework [2]. *Agents* are active, social, and adaptable software system entities situated in some environment and capable of autonomous execution of actions in order to achieve their objectives [28]. Furthermore, most problems are too complex to be solved by just one agent — one must create a multiagent system (MAS) with several agents working together to achieve their objectives and ultimately deliver the desired application. Therefore, adopting the agent-oriented approach to software engineering means that the problem is decomposed into multiple, autonomous, interacting agents, each with their own objectives. Agents in MAS frequently represent individuals, companies, etc. This means that there is an "underlying organizational context" [8] in MAS. Like humans, agents need to coordinate their activities, cooperate, request help from others, etc., often through negotiation. Unlike in object-oriented or component-based systems, interactions in multiagent systems occur through high-level agent communication languages, so interactions are mostly viewed not at the syntactic level, but "at the knowledge level, in terms of goal delegation, etc." [8]. Therefore, modeling and analyzing agents' mental states helps in the specification and analysis of multiagent systems.

In requirements engineering (RE), goal-oriented approaches, e.g., KAOS [4] have become prominent. In Goal-Oriented Requirements Engineering (GORE), high-level stakeholder objectives are identified as goals and later refined into fine-grained requirements assignable to agents/components in the system-to-be or in its environment. Their reliance on goals makes goal-oriented requirements engineering methods and agent-oriented software engineering a great match. Moreover, agent-oriented analysis is central to requirements engineering since the assignment of responsibilities for goals and constraints among components in the software-to-be and agents in the environment is the main outcome of the RE process [10]. Therefore, it is natural to use a goal-oriented requirements engineering approach when developing MAS. With GORE, it is easy to make the transition from the requirements to the high-level MAS specifications. For example, strategic relationships among agents will become high-level patterns of inter-agent communication.

Thus, it would be desirable to devise an agent-oriented requirements engineering approach with a formal component that supports rigorous formal analysis, including reasoning about agents' goals (and knowledge). This would allow for rigorous formal analysis of the requirements expressed as the objectives of the agents in a MAS.

Ordinary ConGolog does not support the specification of the intentional features of *i** models, that is, the *mental states* of the agents in the system/organization modeled; these must be operationalized before they are mapped into ConGolog. But there is an extension of ConGolog called the *Cognitive Agents Specification Language* (*CASL*) [22, 23, 24] that supports formal modeling of agent mental states, incomplete agent knowledge, etc. Mapping *i** models into CASL gives the modeler the flexibility and intuitiveness of the *i** notation as well as the powerful formal analysis capabilities of CASL. So we will extend the *i**-ConGolog approach to combine *i** with CASL and accommodate formal models of agents' mental states. Our intermediate notation will be generalized to support the intentional/mental state modeling features of CASL [11, 12], in what we will call *intentional annotated SR (iASR) diagrams*. With our *i**-CASL-based approach, a CASL model can be used both as a requirements analysis tool and as a formal high-level specification for a multiagent system that satisfies the requirements. This model can be formally analyzed using the CASLve [22, 24] verification tool or other tools and the results can be fed back into the requirements model.

One of the main features of this approach is that goals (and knowledge) are assigned to particular agents thus becoming their subjective attributes as opposed to being objective system properties as in many other approaches, e.g., Tropos [2] and KAOS [4]. This allows for the modeling of conflicting goals, agent negotiation, information exchange, complex agent interaction protocols, etc.

The rest of the chapter is organized as follows. Section 2 briefly introduces *i** and a case study that we will refer to throughout the chapter, and gives an overview of the ConGolog framework. Section 3 presents our approach to map *i** diagrams into ConGolog formal specifications and discusses the use of simulation to validate the models. Section 4 discusses our second approach where *i** models are mapped into CASL, to preserve the intentional features of the models in the formal specifications; we also discuss verification. We conclude in Section 5 by summarizing our results, comparing our approach to related work, and discussing possible extensions.

2 Background

2.1 The *i** Framework and a Case Study

*i** [29] is an agent-oriented modeling framework that can be used for requirements engineering, business process reengineering, etc. *i** centers on the notion of *intentional actor* and *intentional dependency*. In the approaches described here, we use *i** as a graphical requirements modeling notation. We will assume a basic knowledge of *i** in the remainder; to learn about *i** see [30] or the chapter by Yu in this book. We will add various new notational elements to SR diagrams to produce our ASR and iASR diagrams; we will discuss these in detail in later sections. Note also that we do not use softgoals or resource dependencies in ASR and iASR (we will explain why later).

To illustrate the approach that we propose, we will use a variant of the meeting scheduling problem, which has become a popular exemplar in RE [9]. In the context of the *i** modeling framework a meeting scheduling process was first analyzed in [30]. We introduce a number of modifications to the meeting scheduling process to make our models easier to understand. For instance, we take the length of meetings to be the whole day. We also assume that in the environment of the system-to-be there is a legacy software system called the Meeting Room Booking System (MRBS) that handles the booking of meeting rooms. Complete case studies are presented in [11, 12].

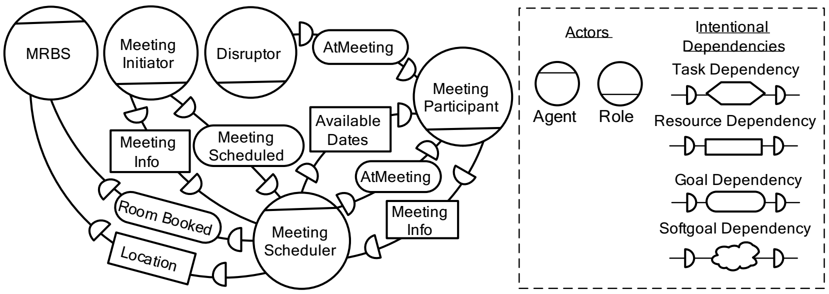


Fig. 1. The Meeting Scheduler in its environment

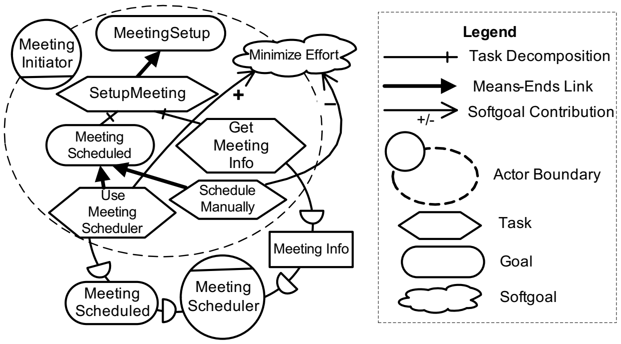


Fig. 2. SR model for the meeting initiator

Fig. 1 is a Strategic Dependency diagram showing the computerized Meeting Scheduler (MS) agent in its environment. Here, the role Meeting Initiator (MI) depends on the MS for scheduling meetings and for being informed about the meeting details. The MS, in turn, depends on the Meeting Participant (MP) role for attending meetings and for providing his/her available dates to it. The MS uses the booking system to book rooms for meetings. The Disruptor actor represents outside actors that cause changes in participants' schedules, thus modeling the environment dynamics.

Fig. 2 is a simple SR models showing some details of the MI process. To schedule meetings, the MI can either do it manually, or delegate it to the scheduler. Softgoal contribution links specify how process alternatives affect quality requirements (softgoals), and so softgoals such as `MinimizeEffort` in Fig. 2 are used to evaluate these alternatives.

2.2 The Formal Foundations: The Situation Calculus and ConGolog

ConGolog [5] is a framework for process modeling and agent programming. It is based on the situation calculus [15], a language of predicate logic for representing dynamically changing worlds. The ConGolog framework can be used to model *complex processes* involving loops, concurrency, multiple agents, etc. Because it is logic-based, the framework can accommodate incompletely specified models, either in the sense that the initial state of the system is not completely specified, or that the processes involved are non-deterministic and may evolve in any number of ways.

A ConGolog specification includes two components. First, to support reasoning about the processes executing in a certain domain, that domain must be formally specified: what predicates describe the domain, what primitive actions are available to agents, what the preconditions and effects of these actions are, and what is known about the initial state of the system. The other component of a ConGolog specification is the model of the process of interest, i.e. the behaviour of the agents in the domain.

In ConGolog and in the situation calculus, a dynamic domain is modeled in terms of the following entities:

- *Primitive actions*: all changes to the world are assumed to be the result of named primitive actions that are performed by some agent; primitive actions are represented by terms, e.g. *acceptAgreementReq(participant,MS,reqID, date)*, i.e. the *participant* agent accepts the request *reqID* from the *MS* agent to attend a meeting on *date*.
- *Situations*: these correspond to possible world histories viewed as sequences of actions. The actual initial situation (where no actions have yet been executed) is represented by the constant S_0. There is a distinguished binary function symbol *do* and a term *do(a,s)* denotes the situation that results from action *a* being performed in situation *s*. For example, $do(a_3,do(a_2,do(a_1,S_0)))$ represents the situation where first a_1, then a_2, and then a_3 have been performed starting in the initial situation S_0. Thus, situations are organized in tree structures rooted in some initial situation; the situations are nodes in the tree and the edges correspond to primitive actions.
- *Fluents*: these are properties, relations, or functions of interest whose value may change from situation to situation; they are represented by predicate and function symbols that take a situation term as their last argument, e.g. *agreementReqRcvd(participant,MS,reqID, date,s)*, i.e. *participant* has received a request *reqID* from *MS* to agree to hold a meeting on *date* in situation *s*. Non-fluent predicates/functions may also be used to represent static features of the domain.

The dynamics of a domain are specified using four kinds of axioms:

- *Action precondition axioms*: these state the conditions under which an action can be performed; they use the predicate *Poss(a,s)*, meaning that action *a* is possible in situation *s*. E.g., in our meeting scheduling domain, we have:

$$\begin{aligned}&Poss(acceptAgreementReq(participant,MS,reqID,date),s) \equiv\\ &\quad agreementReqRcvd(participant,MS,reqID,date,s) \wedge\\ &\quad dateFree(participant,date,s)\end{aligned}$$

  This says that in situation *s*, *participant* may perform the action of accepting a request *reqID* from *MS* to hold a meeting on *date* if and only if he has received a request to that effect and the date is free for him.

- *Successor state axioms (SSA)*: these specify how the fluents are affected by the actions in the domain. E.g., in our meeting scheduling domain, we have:

$$\begin{aligned}&agreementReqRcvd(participant,MS,reqID,date,do(a,s)) \equiv\\ &\quad a = requestAgreement(MS,participant,date) \wedge\\ &\quad requestCounter(s) = reqID \vee\\ &\quad agreementReqRcvd(participant,MS,reqID,date,s)\end{aligned}$$

  This says that *participant* has received a request *reqID* from *MS* to agree to hold a meeting on *date* in situation *do(a,s)* if and only if the action *a* is such a request and the value of the request counter is *reqID* or if she had already received such a request in situation *s*.

  Successor state axioms were introduced by Reiter [19] and provide a solution to the frame problem. They can be generated automatically from a specification of the effects of primitive actions if we assume that the specification is complete. Lespérance et al. [13] described a convenient high-level notation for specifying the effects (and preconditions) of actions and a tool that compiles such specifications into successor state axioms.

- *Initial situation axioms*: these specify the initial state of the modeled system. E.g., in our meeting scheduling domain, we might have the following initial situation axiom: $participantTimeSchedule(Yves,S_0) = [10,12]$, representing the fact that agent *Yves* is busy on the 10th and 12th in the initial situation.
- *Other axioms*: these include unique name axioms for actions, axioms specifying the agent of each type of action, and domain independent foundational axioms as described in [19].

The process of a system is specified procedurally in the ConGolog framework. We define a *main* procedure that specifies the behaviour of the whole system. Every agent has an associated ConGolog procedure to represent its behaviour in the system. The behaviour of agents is specified using a rich high-level programming language with recursive procedures, while loops, conditionals, non-determinism, concurrency, and interrupts [5]. The available constructs include:

a,	primitive action
$\varphi?$,	wait for condition
$\delta_1;\delta_2$,	sequence
$\delta_1 \vert \delta_2$,	nondeterministic branch
δ^*,	nondeterministic iteration
$\pi v.\delta$,	nondeterministic choice of argument
if φ ***then*** δ_1 ***else*** δ_2 ***endIf***,	conditional
while φ ***do*** δ ***endWhile***,	while loop
$\delta_1 \Vert \delta_2$,	concurrency with equal priority
$\delta_1 \rangle\rangle \delta_2$,	concurrency with *δ1* at higher priority
guard φ ***do*** δ ***endGuard***	guard
<v: $\varphi \rightarrow \delta$ **until** α>	interrupt
$\beta(p)$,	procedure call

Note the presence of several non-deterministic constructs. For instance, *δ1|δ2* nondeterministically chooses between executing *δ1* or *δ2*. *πv.δ* non-deterministically picks a binding for the variable *v* and performs the program *δ* for that binding. *δ** performs *δ* zero or more times. A test/wait action *φ?* blocks until the condition φ becomes true. *<v: φ → δ **until** α>* represents an interrupt; when the trigger condition *φ* becomes true for some value of *v*, the interrupt triggers and the body, *δ*, is executed; the interrupt may trigger repeatedly as long as its cancellation condition *α* does not hold. The guard construct blocks the execution of a program *δ* until the condition *φ* becomes true.

A formal semantics based on transition systems (structural operational semantics) has been specified for ConGolog [5]. It defines a special predicate ***Do****(program,s,s')* that holds if there is a successful execution of *program* that ends in situation *s'* after starting in *s*. Communication between agents can be represented by actions performed by the sender agent, which affect certain fluents that the recipient agent has access to.

A process simulation and validation tool for ConGolog has been implemented [5]. It uses an interpreter for ConGolog implemented in Prolog. This implementation requires that the precondition axioms, successor state axioms, and axioms about the initial situation be expressed as Prolog clauses, and relies on Prolog's closed world assumption and negation as failure. Thus with this tool, simulation can only be performed for completely specified initial states.

A verification tool has also been developed [22, 24]. We discuss verification in Section 4.3. De Giacomo et al. [5] describe applications of ConGolog in different areas, such as robot programming, personal assistants, etc. Lespérance et al. [13] discuss the use of ConGolog (without combining it with *i**) for process modeling and requirements engineering.

3 Using ConGolog for the Analysis, Simulation, and Verification of *i** Models

While the informal *i** notation can be successfully used for modeling and analysing relatively small systems, formal analysis is very helpful with larger systems. Thus, formal analysis of *i** models is one of the goals of the approaches presented here. Another aim is to allow for a smooth transition from requirements specifications to high-level design for agent-based systems. While the *i** SR diagram notation allows many aspects of processes to be represented, it is somewhat imprecise and the models produced are often incomplete. For instance, it is not specified whether the subtask in a task decomposition link has to be performed once or several times. In a ConGolog model, on the other hand, the process must be completely and precisely specified (although non-deterministic processes are allowed). We need to bridge this gap. To do this, we will introduce a set of *annotations* to SR diagrams that allow the missing information to be specified. We also want to have a tight mapping between this *Annotated SR (ASR) diagram* and the associated ConGolog model, one that specifies which parts of each model are related. This allows us to identify which parts of the ConGolog model need to be changed when the SR/ASR diagram is modified and vice versa. The *i**-ConGolog approach that we describe in this section is largely based on [26, 27].

3.1 Annotated SR Diagrams

The main tool that we use for disambiguating SR diagrams is *annotations*. Annotations allow analysts to model the domain more precisely and capture data/control dependencies among goals and other details. Annotations, introduced in [26, 27] and extended in [11, 12], are textual constraints on ASR diagrams and can be of three types: composition, link, and applicability conditions. *Composition annotations* (specified by σ in Fig. 3) are applied to task and means-ends decompositions and specify how the subtasks/subgoals are to be combined to execute the supertask and achieve the goal respectively. Four types of composition are allowed: sequence (";"), which is the default for task decompositions, concurrency ("||"), prioritized concurrency ("»"), and alternative ("|"), which is the default for means-ends decompositions. These annotations are applied to subtasks/subgoals from left to right. E.g., in Fig. 3, if the "»" annotation is applied, n_1 has the highest priority, while n_k has the lowest. The choice of composition annotations is based on the ways actions and procedures can be composed in ConGolog.

Link annotations (γ_i in Fig. 3) are applied to subtasks/subgoals (n_i) and specify how/under which condition they are supposed to be achieved/executed. There are six types of link annotations (corresponding to ConGolog operators): *while* loop, *for* loop (introduced in [22]), the *if* condition, the pick, the interrupt, and the guard (introduced in [11, 12]). The difference between the *if* annotation and the guard is that the guard blocks execution until its condition becomes true while the task with the *if* link annotation is skipped if the condition is not true. The pick annotation (*π(VariableList,Condition)*) non-deterministically picks values for variables in the

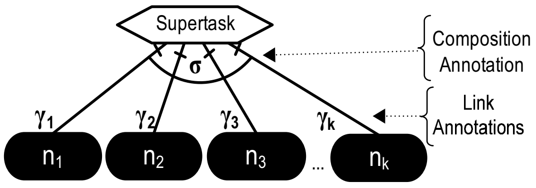


Fig. 3. Composition and link annotations

subtask that satisfy the condition. The interrupt (*whenever(varList, Condition, CancelCondition)*) fires and executes the subtask whenever there is a binding for the variables that satisfies the condition until the cancellation condition becomes true. Guards (*guard(Condition)*) block the subtask's execution until the condition becomes true. The absence of a link annotation on a particular decomposition link indicates the absence of any conditions on the subgoal/subtask.

If alternative means of achieving a certain goal exist, the designer can specify under which circumstances it makes sense to try each alternative. We call these *applicability conditions* and introduce a new annotation *ac(condition)* to be used with means-ends links to specify them. The presence of an applicability condition (AC) annotation specifies that only when the condition is true may the agent select the associated alternative in attempting to achieve the parent goal. E.g., one may specify that phoning participants to notify them of the meeting details is applicable only for important participants, while the email option is applicable for everyone (see Fig. 6). When there is no applicability condition, an alternative can always be selected.

3.2 Increasing Precision with ASR Models

The starting point for developing an ASR diagram for an actor is the regular SR diagram for that actor (e.g., see Fig. 2). It then can be appropriately transformed to become an ASR diagram every element of which can easily be mapped into ConGolog. The steps for producing ASR diagrams from SR ones include the addition of model annotations, the removal of softgoals, the deidealization of goals [9], and the addition of details of agent interaction to the model. Since an ASR diagram is going to be mapped into a ConGolog specification consisting of parameterized procedures, parameters for annotations/goals/tasks capturing the details of events as well as what data or resources are needed for goal achievement or task execution can be specified in ASR diagrams (see Fig. 6) to simplify the generation of ConGolog code. However, we sometimes omit the parameters in ASR diagrams for brevity.

Softgoals. Softgoals represent non-functional requirements [3] and are imprecise and difficult to handle in a formal specifications language such as ConGolog. Therefore in this approach, we use softgoals to choose the best process alternatives and then remove them before ASR diagrams are produced. Alternatively, softgoals can be operationalized or metricized, thus becoming hard goals. The removal of softgoals in ASR diagrams is a significant deviation from the standard *i** framework.

Deidealization of goals. Goals in ASR diagrams that cannot always be achieved are replaced by weaker goals that can. This involves identifying various possible failure conditions and guarding against them.

*Providing agent interaction details. i** usually abstracts from modeling any details of agent interactions. In ASR diagrams, we specify the interactions through which intentional dependencies are realized by the actors involved. Interactions are specified as processes involving various "communication" primitive actions that change the state of the system. The effects of these actions are modeled using ordinary fluents/properties. This supports simulation, but does not capture the fact that these actions operate on the mental states of the communicating agents. We address this in Section 4. Agent interaction details include tasks such as requests for services or information from agents in the system, tasks that supply information or communicate about success or failure in providing services, etc. Arbitrarily complex interaction protocols can be specified. We assume that the communication links are reliable.

In ASR diagrams, all resource dependencies are modeled more precisely using either goal or task dependencies according to the level of freedom that the dependee has in supplying the resource.

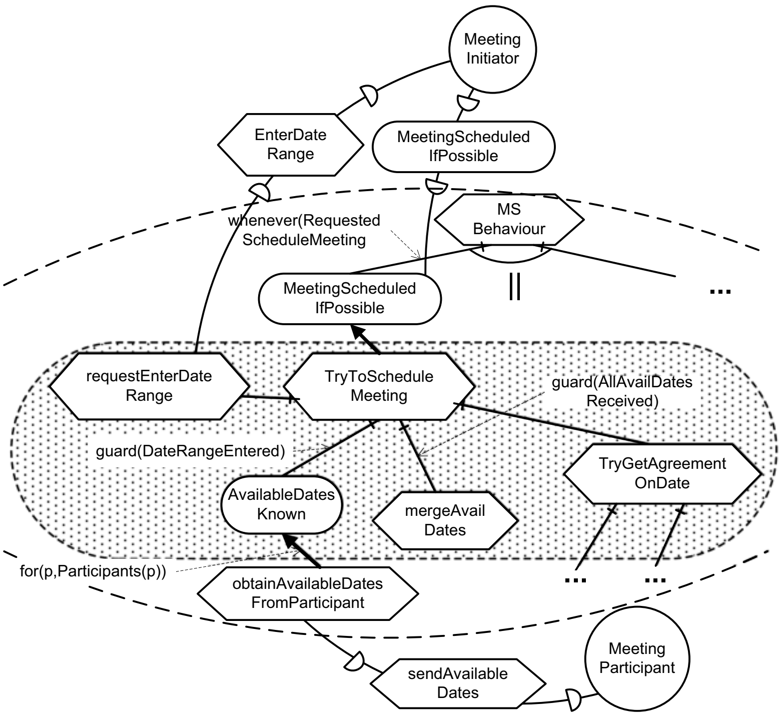


Fig. 4. A fragment of the ASR diagram for the MS agent

Fig. 4 shows a small fragment of the ASR diagram for the Meeting Scheduler agent. This model shows a very high-level view of the achievement of the goal TryToScheduleMeeting. Here, the MS must get the suggested meeting dates from the MI, get the available dates from the participants, find agreeable dates (potential dates for the meeting), and try to arrange the meeting on one of those days. Various annotations have been added to the model. The absence of a composition annotation for the TryToScheduleMeeting task indicates that it is sequentially decomposed. There are interrupt/guard annotations that let the MS agent monitor for incoming requests and for replies to its queries about the meeting date range and available dates for participants. The *for* annotation indicates that the querying for the available dates is iterated for all the participants. Note that the goal TryToScheduleMeeting in Fig. 4 is a deidealized (weakened) goal.

3.3 Mapping ASR Diagrams into ConGolog

Once all necessary details have been introduced into an ASR diagram, it can be mapped into a corresponding formal ConGolog model, thus making the model amenable to formal analysis. The modeler must define a *mapping* ***m*** that maps every element (except for intentional dependencies) of an ASR diagram into ConGolog. This mapping associates ASR diagram elements with ConGolog procedures, primitive actions, and formulas so that a ConGolog program can be generated from an ASR diagram. Specifically, agents are mapped into constants that serve as their names and ConGolog procedures that specify their behaviour; roles and positions are mapped into similar procedures with an agent parameter so that they can be instantiated by individual agents. So, when an agent plays several roles or occupies several positions, it executes the procedures that correspond to these roles/positions concurrently. Leaf-level task nodes are mapped into ConGolog procedures or primitive actions. Composition and link annotations are mapped into the corresponding ConGolog operators, and conditions present in the annotations map into ConGolog formulas.

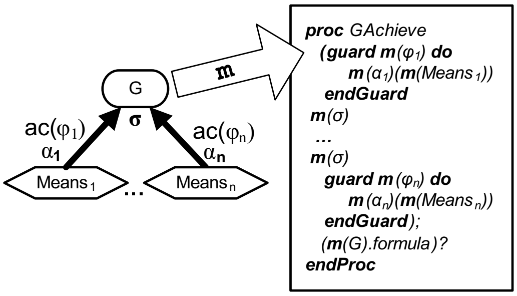
Mapping Task Nodes. A non-leaf task node with its decomposition is automatically mapped into a ConGolog procedure that reflects the structure of the decomposition and all the annotations.

Consider the shaded part of Fig. 4, where the task TryToScheduleMeeting is decomposed into a number of subtasks/subgoals. This task will be mapped into the following ConGolog procedure (it contains parts still to be mapped into ConGolog; they are the parameters of the mapping ***m***). Here, the parameter *mid* stands for "meeting ID", a unique meeting identifier:

```
proc TryToScheduleMeeting(mid,mi)
  requestEnterDateRange(MS,mi,mid);
  guard m(DateRangeEntered) do
    m(AvailableDatesKnown).achieve;
  endGuard;
  guard m(AllAvailDatesReceived) do
    mergeAvailDates(MS,mid);
  endGuard;
  TryToGetAgreementOnDate(MS,mid);
endProc
```

Notice that the mapping of tasks into ConGolog procedures is compositional. We have defined a set of mapping rules that formally specify this part of the mapping process.

Mapping Goal Nodes. In the *i**-ConGolog approach, goal nodes are mapped into a ConGolog formula that represents the desired state of affairs associated with the goal and a procedure that encodes means for achieving the goal. The achievement procedure is generated from the decomposition of the goal into means for achieving it, which is modeled in the ASR diagram through means-ends links. This is similar to the mapping of task decompositions as seen above and can be performed automatically. The achievement procedure for a goal *G* can be referenced as ***m****(G).achieve* (e.g., see the code fragment above). Fig. 5 shows a generic goal decomposition together with the generated achievement procedure. At the end of the achievement procedure, there is typically a test that makes sure that the goal is achieved: ***m****(G).formula)?*.



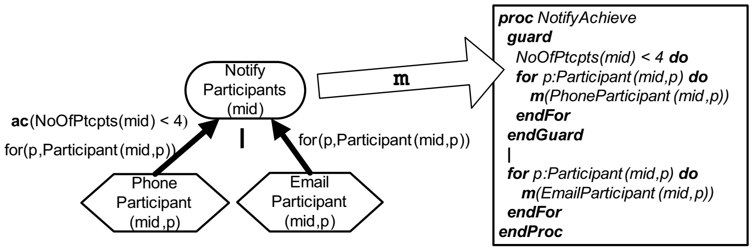
```
proc GAchieve
  (guard m(φ1) do
      m(α1)(m(Means1))
  endGuard
 m(σ)
 ...
 m(σ)
  guard m(φn) do
      m(αn)(m(Meansn))
  endGuard);
  (m(G).formula)?
endProc
```

Fig. 5. Generating a goal achievement procedure

The default composition annotation for means-ends decompositions (represented by σ in Fig. 5) is alternative ("|"). This indicates that the means for achieving the goal is selected non-deterministically. As shown in Fig. 5, each goal achievement alternative is wrapped in a guard operator with the guard condition being the result of mapping the corresponding applicability condition annotation. This ensures that an alternative will only be selected when it can begin execution and its applicability condition holds. Other composition annotations (e.g. concurrency or sequence) can also be used. Note that neither ConGolog nor CASL currently provides built-in language constructs for sophisticated handling of alternative selection, execution monitoring, failure handling, retries, etc.; this is an area for future work.

Since in this approach, softgoals are removed from ASR diagrams, applicability conditions can be used to capture in a formal way the fitness of the alternatives with respect to softgoals (this fitness is normally encoded by the softgoal contribution links in SR diagrams). For example, one can specify that phoning participants to notify them of the meeting details is applicable only in cases with few participants, while the email option is applicable for any number of participants (see Fig. 6). This may be due to the softgoal Minimize Effort that has been removed from the model before the ASR diagram was produced.

In addition to applicability conditions, other link annotations can be used with means-ends decompositions to specify extra control information. These are represented by α_i in Fig. 5 and are exemplified by the *for* loop annotations in Fig. 6. Note that these annotations are applied after applicability conditions.


Notify Participants (mid)
ac(NoOfPtcpts(mid) < 4)
for(p,Participant(mid,p))
|
for(p,Participant(mid,p))
Phone Participant (mid,p)
Email Participant (mid,p)
m

```
proc NotifyAchieve
 guard
  NoOfPtcpts(mid) < 4 do
  for p:Participant(mid,p) do
     m(PhoneParticipant(mid,p))
  endFor
 endGuard
 |
 for p:Participant(mid,p) do
    m(EmailParticipant(mid,p))
 endFor
endProc
```

Fig. 6. Goal achievement procedure example

Specifying Domain Dynamics. To obtain a complete ConGolog specification, one needs to provide the declarative part of the specification, namely an action precondition axiom for every primitive action, a successor state axiom for every fluent, and initial state axioms, as described in Section 2.2.

3.4 Simulation

ConGolog models can be executed to run process simulation experiments. To do this, the modeler must first specify an instance of the overall system. We do this by defining a `main` procedure. Here is how this looks in the ConGolog simulation tool notation (`#=` is the concurrent execution operator):

```
proc(main,[
     initiator_behavior(mi,ms)#=
     meetingScheduler_behavior(ms,mi)#=
     participant_behaviour(yves,ms)#=
     participant_behaviour(alexei,ms)#=
]).
```

Here, there are the Meeting Initiator agent, `mi`, the Meeting Scheduler `ms`, and two participants, `yves` and `alexei`. The modeler must also provide a *complete* specification of the initial state of the system. Here, the possible meeting dates are represented as integers in order to simplify the explanation. Initially the schedule for the participant `alexei` is `[11,12,14]`, i.e., `alexei` is busy on the 11th, 12th, and 14th of some month. The schedule for the participant `yves` is `[10,12]`, i.e. `yves` is busy on the 10th and 12th. The Meeting Initiator `mi` wants to schedule a meeting with `alexei` and `yves` on the 12th or 14th. Then the modeler can execute the `main` procedure to obtain a simulation trace. The simulation obtained from this instance of the system is as follows:

```
// start interrupts in initial situation
startInterrupts
// mi requests ms to schedule a meeting with alexei and yves
requestScheduleMeeting(mi,ms,[alexei,yves])
// ms requests mi to enter the possible date range for meeting with id = 1
requestEnterDateRange(ms,mi,1)
// mi enters 12, 14 as possible meeting dates
enterDateRange(mi,ms,1,[12,14])
// ms requests available dates from all participants
obtainAvailDatesFromParticipant(ms,yves,1)
obtainAvailDatesFromParticipant(ms,alexei,1)
// yves sends his available dates
sendAvailDates(yves,ms,1,[…])
// alexei sends his available dates
sendAvailDates(alexei,ms,1,[…])
mergeAvailableDates(ms,1)
// ms finds the list of common available dates empty
setAllMergedlist(ms,1,[])
```

```
// ms notifies both participants and the initiator that it failed to schedule
// meeting 1
notifyFail(ms,mi,1,[alexei,yves])
notifyFail(ms,alexei,1,[alexei,yves])
notifyFail(ms,yves,1,[alexei,yves])
```

Generally, this validation step of the process involves finding gaps or errors in the specification by simulating the processes. The ConGolog code can be instrumented with tests (using the *"?"* operator) to verify that desired properties hold, e.g., during or at the end of the execution of the program. Alternative specifications can be also compared. A graphical user interface tool for conducting such simulation experiments is available, see [13]. As mentioned, the simulation tool requires a *complete* specification of the initial state. This limitation comes from the fact that the tool uses Prolog and its closed world assumption to reason about how the state changes. The tool (like ConGolog itself) does not provide support for modeling agent mental states and how they are affected by communication and other actions. As we saw in the examples, it is possible to model limited aspects of this using ordinary actions and fluent predicates, but this does not capture the full logic of mental states and communication. Work is underway to relax these limitations and develop techniques for efficient reasoning about limited types of incomplete knowledge and knowledge-producing actions in ConGolog [20]. ConGolog models can also be verified using the CASLve tool discussed in Section 4.3.

4 Modeling Mental States in Requirements Engineering

4.1 Motivation

Suppose that we are employing an approach like Tropos [2, 6] to model a simple goal delegation involving two agents. Fig. 7 shows a goal dependency where the Meeting Scheduler depends on the Meeting Participant for attending a meeting. We would like to

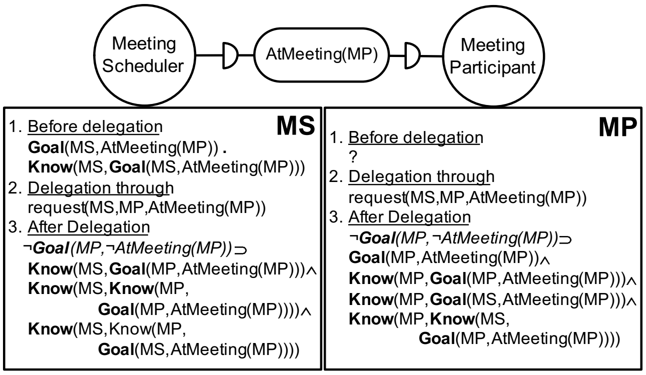

Meeting Scheduler
AtMeeting(MP)
Meeting Participant
MS
1. Before delegation
Goal(MS,AtMeeting(MP)) .
Know(MS,Goal(MS,AtMeeting(MP)))
2. Delegation through
request(MS,MP,AtMeeting(MP))
3. After Delegation
¬Goal(MP,¬AtMeeting(MP))⊃
Know(MS,Goal(MP,AtMeeting(MP)))∧
Know(MS,Know(MP, Goal(MP,AtMeeting(MP))))∧
Know(MS,Know(MP, Goal(MS,AtMeeting(MP))))
MP
1. Before delegation
?
2. Delegation through
request(MS,MP,AtMeeting(MP))
3. After Delegation
¬Goal(MP,¬AtMeeting(MP))⊃
Goal(MP,AtMeeting(MP))∧
Know(MP,Goal(MP,AtMeeting(MP)))∧
Know(MP,Goal(MS,AtMeeting(MP)))∧
Know(MP,Know(MS, Goal(MP,AtMeeting(MP))))

Fig. 7. A motivating example

be able to analyze this interaction and predict how it will affect the agents' goals and knowledge. Using the *i**-CASL approach presented in this section [11, 12], one can create a formal model based on the diagram, analyze it, and conclude that, e.g., before the goal delegation, the MS has the goal AtMeeting(MP) and knows about this fact. After the delegation (and provided that the MP did not have a conflicting goal), the MS knows that the MP has acquired the goal, that the MP knows that it has the goal, and that the MP knows that the MS has the same goal, etc. Similar questions can be asked about MP.

Note that the change in the mental state of the requestee agent is the core of goal delegation. One of the main features of the *i**-CASL approach is that goals (and knowledge) are assigned to particular agents thus becoming their subjective attributes as opposed to being objective system properties as in many other approaches (e.g., [4]). This allows for the modeling of conflicting goals, agent negotiation, information exchange, complex agent interaction protocols, etc. In CASL, the full logic of these mental states and how they change is formalized. The *i**-CASL approach thus allows for creating richer, more expressive specifications with precise modeling of agents' mental states. However, the more complex CASL models currently require the use of a theorem-prover-based verification tool such as CASLve and cannot be used with the ConGolog simulation tool.

4.2 The Cognitive Agents Specification Language

The Cognitive Agents Specification Language (CASL) [22, 23] is a formal specification language that extends ConGolog to incorporate *models of mental states* expressed in the situation calculus [21]. CASL uses modal operators to formally represent agents' knowledge and goals; communication actions are provided to update these mental states and ConGolog is then employed to specify the behaviour of agents. The logical foundations of CASL allow it to be used to specify and analyze a wide variety of MAS as shown in [22, 23]. For instance, it can model non-deterministic behaviours and systems with an incompletely specified initial state. Similar to ConGolog (see Section 2.2), CASL specifications consist of two parts: the model of the domain and its dynamics (the declarative part) and the specification of the agents' behaviour (the procedural part).

The formal representation for both goals and knowledge in CASL is based on a *possible worlds semantics* incorporated into the situation calculus, where situations are viewed as possible worlds [16, 21]. CASL uses accessibility relations K and W to model what an agent knows and what it wants respectively. $K(agt,s',s)$ holds if the situation s' is compatible with what the agent agt knows in situation s. In this case, the situation s' is called *K-accessible*. When an agent does not know the truth value of some formula φ, it considers possible (formally, K-accessible) some situations where φ is true and some where it is false. An agent knows that φ in situation s if φ is true in all its K-accessible situations in s: $\boldsymbol{Know}(agt,\varphi,s) = \forall s'(K(agt,s',s) \supset \varphi[s'])$. Constraints on the K relation ensure that agents have positive and negative introspection (i.e., agents know whether they know/don't know something) and guarantee that what is known is true. Built-in communication actions such as *inform* are used for exchanging information among agents. The precondition for the *inform* action ensures that no false information is transmitted. The changes to agents' knowledge due to communication and other actions are specified by the successor state axiom for the *K* relation. The specification ensures that agents are aware of the execution of all actions. Enhanced accounts of knowledge change and communication in the situation calculus have also been proposed to handle, for instance, encrypted messages [23] or belief revision [25].

The accessibility relation *W(agt,s',s)* holds if in situation *s* an agent considers that everything that it wants to be true actually holds in *s'*, which is called *W*-accessible. We use the formula ***Goal****(agt,ψ,s)* to indicate that in situation *s* the agent *agt* has the goal that *ψ* holds. The definition of ***Goal*** says that *ψ* must be true in all *W*-accessible situations that have a *K*-accessible situation in their past. This ensures that while agents may want something they know is impossible to achieve, the goals of agents must be consistent with what they currently know. There are constraints on the *W* and *K* relations that ensure that agent's goals are consistent and that agents introspect their goals. In our approach, we mostly use achievement goals that specify the desired states of the world. We use the formula ***Goal****(agt,****Eventually****(ψ),s)* to state that *agt* has the goal that *ψ* is eventually true. The built-in communication actions *request* and *cancelRequest* are used by agents to request services from other agents and to cancel such requests respectively. Requests are used to establish intentional dependencies among actors and lead to changes in goals of the requested agent. The dynamics of the *W* relation are specified, as usual, by a successor state axiom that guarantees that no inconsistent goals are adopted.

4.3 The *i**-CASL Notation and Process

Increasing Precision with Intentional Annotated Strategic Rationale Models. Our aim in this approach is to tightly associate *i** models with formal specifications in CASL. As was the case with the *i**-ConGolog approach presented in Section 3, we use an intermediate notation, *Intentional Annotated SR (iASR) diagrams,* to bridge the gap between SR diagrams and CASL specifications.

When developing an iASR diagram, one starts with the corresponding SR diagram (e.g., see Fig. 2). The steps for producing iASR diagrams from the corresponding SR ones are similar to the ones presented in Section 3.

Agent Goals in iASR Models. A CASL agent has procedural (behaviour) and declarative (mental state) components. iASR diagrams only model agent processes and thus are used to represent the procedural component of CASL agents. A goal node in an iASR diagram indicates that the agent knows that the goal is in its mental state and is prepared to deliberate about if and how to achieve it. For the agent to modify its behaviour in response to the changes to its mental state, it must synchronize its procedural and declarative components (see Fig. 8A). Agent mental states usually change as a result of communication acts that realize goal delegation and information exchange. So, the procedural component of the agent must monitor for these changes. The way to do this is to use interrupts or guards with their conditions being the presence of certain goals or knowledge in the mental state of the agent (Fig. 8B). Procedurally, the goal node is interpreted as invoking the means to achieve it.

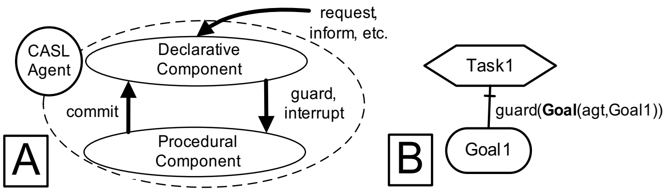


Fig. 8. Synchronizing declarative and procedural components of CASL specifications

In CASL, only communication actions have effects on the mental state of the agents. However, we also would like the agents to be able to change their mental state on their own by executing the action *commit(agent,φ)*, where *φ* is a formula that the agent (or the modeler) wants to hold. Thus, in iASR diagrams all agent goals must be acquired either from intentional dependencies or by using the *commit* action. By introducing goals into the models of agent processes, the modeler captures the fact that multiple existing or potential alternatives exist in these processes and makes sure the mental state of agents reflect this. This allows agents to reason about their goals and ways to attain them at runtime.

Modeling agent interactions. We take an intentional stance towards modeling agent interactions. We are modeling them with built-in generic communication actions (e.g., *request*, *inform*) that modify the mental states of the agents. In iASR models, these generic communication actions are used to request services, provide information, etc. Also, the conditions in annotations and communication actions (as well as the *commit* action) may refer to the agents' mental states, knowledge and goals. Because of the importance of agent interactions in MAS, in order to formally verify multiagent system specifications in CASL, all high-level aspects of agent interaction must be provided in the corresponding iASR models.

Fig. 9A and Fig. 9B illustrate how an intentional goal dependency RoomBooked (see Fig. 1) can be modeled in SR and iASR models respectively. It is established by the MS's execution of the *request* action (with that goal as the argument) towards the MRBS agent. This will cause the MRBS to acquire the goal RoomBooked (if it is consistent with its existing goals). The interrupt in the iASR model for the MRBS monitors

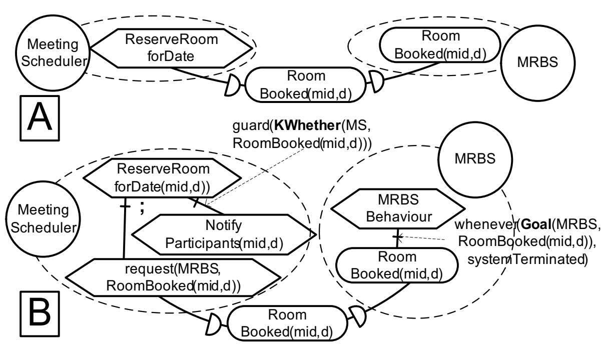

Meeting Scheduler
ReserveRoom forDate
Room Booked(mid,d)
Room Booked(mid,d)
MRBS
A
guard(**KWhether**(MS, RoomBooked(mid,d)))
MRBS
Meeting Scheduler
ReserveRoom forDate(mid,d))
;
Notify Participants(mid,d)
request(MRBS, RoomBooked(mid,d))
MRBS Behaviour
whenever(**Goal**(MRBS, RoomBooked(mid,d)), systemTerminated)
Room Booked(mid,d)
Room Booked(mid,d)
B

Fig. 9. Adding iASR-level agent interaction details

its mental state for the goal and triggers the behaviour for achieving it (i.e. booking a room, which is not shown) when the goal is acquired. Also, once the MS's knowledge state is updated and it knows whether (formally, ***KWhether***) the room has been booked (note the guard condition), the task for notifying participants will be triggered.

From iASR Models to CASL Specifications. Once an iASR model has been produced, it can be mapped into a CASL specification for formal analysis.

As previously, the modeler defines a mapping ***m*** that associates iASR model elements (except for dependencies) with CASL procedures, primitive actions, and formulas, so that a CASL program can be generated from an iASR model. Specifically, actors are mapped into CASL procedures, leaf-level tasks are mapped into procedures or primitive actions, while annotations are mapped into CASL operators. Conditions in the annotations map into CASL formulas that can refer to agents' mental states.

Mapping Goal Nodes. An iASR goal node is mapped into a CASL formula (the formal definition for the goal) and an achievement procedure that is based on the means-ends decomposition for the goal in the iASR diagram (see Fig. 5). E.g., a formal definition for *MeetingScheduled(mid,s)* could be: $\exists d[AgreeableDate(mid, date,s) \wedge AllAccepted(mid,date,s) \wedge RoomBooked(mid,date, s)]$. This says that there must be a date agreeable for everybody on which a room was booked and all participants accepted to meet. Often, an initial goal definition is too ideal and needs to be *deidealized* [9] or weakened. See [12] for an example.

CASL's support for reasoning about agents' goals gave us the ability not to maintain meeting participants' schedules explicitly. Rather, we relied on the presence of goals *AtMeeting(participant,mid,date,s)* in their mental states together with an axiom that made sure that they could only attend one meeting per time slot (see [12]).

The achievement procedures for goals are automatically constructed based on the modeled means for achieving them as described in Section 3.

Modeling Dependencies. Intentional dependencies are not mapped into CASL per se — they are established by the associated agent interactions. iASR tasks requesting help from agents will generally be mapped into actions of the type *request(FromAgt,ToAgt,**Eventually**(φ))* for an achievement goal *φ*. For task dependencies, we use *request(FromAgt, ToAgt, **DoAL**(SomeProcedure))* to request that a known procedure be executed while allowing other actions to occur (***DoAL*** stands for "do at least").

In order for a dependency to be established, we also need a commitment from a dependee agent to act on the request from the depender. It must monitor its mental state for the newly acquired goals, which is done using interrupts that trigger whenever unachieved goals of certain types are in their mental states. The bodies of interrupts specify appropriate responses to the messages. Also, cancellation conditions in interrupts allow the agents to monitor for certain requests/informs only in particular contexts (e.g., while some interaction protocol is being enacted). For details, see [11, 12].

Analysis and Verification. Once an iASR model is mapped into the corresponding CASL specification, it is ready to be formally analyzed. One tool that can be used is CASLve [24, 22], a theorem-prover-based verification environment for CASL. CASLve provides a library of theories for representing CASL specifications and lemmas that facilitate various types of verification proofs. In addition to physical executability of agent programs, one can also check for the epistemic feasibility of agent plans [14], i.e., whether agents have enough knowledge to successfully execute their processes. Alternative verification approaches based, for instance, on simulation or model checking can be used. However, they require much less expressive languages, so CASL specifications need to be simplified for these approaches.

If expected properties of the system are not entailed by the CASL model, it means that the model is incorrect and needs to be fixed. The source of an error found during verification can usually be traced to a portion of the CASL code, and to a part of its iASR model, since our systematic mapping supports traceability.

5 Discussion and Future Work

In this chapter, we have presented an approach to requirements engineering that involves the combined use of *i** and some multiagent system specification formalisms, ConGolog and its extension CASL. This allows the requirements engineer to exploit the complementary features of the frameworks. The *i** framework can be used to model social dependencies between agents, perform an analysis of opportunities and vulnerabilities, explore alternatives and trade-offs. These models are then gradually made more precise with the use of *annotated models*. ConGolog or CASL can then be used to model complex processes formally with subsequent verification or simulation (for ConGolog only). Additionally, CASL supports the explicit modeling of agent mental states and reasoning about them. In our approach, both graphical/informal and textual/formal notations are used, which supports a progressive specification process and helps in communicating with the clients, while providing traceability.

There have been a few other proposals for using *i** with formal specification languages for RE. The Trust-Confidence-Distrust (TCD) approach combining *i** and ConGolog to model/analyze trust in social networks was proposed in [7]. TCD is focused on a specific type of applications and has an extended SR notation that is quite different from our proposal in terms sequencing of elements, explicit preconditions, etc.

Formal Tropos (FT) [6] is another approach that supports formal analysis of *i** models though model checking. Its specifications use temporal logic and it can be used at the SD level, unlike our approaches, which use procedural notations that are more suitable for SR models. Unlike CASL, the formal components of FT and the *i**-ConGolog approach do not support reasoning about goals and knowledge and thus require that goals be abstracted out of the specifications. However, most agent interactions involve knowledge exchange and goal delegation. The ability of CASL to formally model and reason about mental states as properties of agents is important and supports new types of analysis (e.g., of conflicting goals).

In future work, we would like to develop tool support for representing ASR/iASR diagrams and mapping them into ConGolog/CASL and for supporting the co-evolution of the two representations. We expect that our RE toolkit will be able to significantly simplify the specification of the declarative component of ConGolog/CASL models. We plan to explore how different types of agent goals (e.g., maintenance) as well as privacy, security, and trust can be handled in CASL. There are also a number of limitations of CASL's formalization of mental state change and communication that should be addressed in future work. One such limitation is that agents cannot send false information. Removing this limitation requires modeling belief revision, which adds a lot of complexity (see [25]). However, this will support modeling of, e.g., malicious and untruthful agents.

We also note that CASL assumes that all agents are aware of all actions being executed in the system. Often, it would be useful to lift this restriction, but dealing with the resulting lack of knowledge about agents' mental states can be challenging.

Finally, there is also ongoing work on supporting limited forms of incomplete knowledge and information acquisition actions in a logic programming-based ConGolog implementation [20]. This may eventually lead to an executable version of CASL where simulation can be performed on models of agents with mental states.

Bibliography

1. Bordini, R.H., Dastani, M., Dix, J., El Fallah Seghrouchni, A. (eds.): Multi-agent programming: Languages, platforms and applications. Springer, Heidelberg (2005)
2. Bresciani, P., Giorgini, P., Giunchiglia, F., Mylopoulos, J., Perini, A.: TROPOS: An agent-oriented software development methodology. Journal of Autonomous Agents and Multi-Agent Systems 8(3), 203–236 (2004)
3. Chung, L., Nixon, B., Yu, E., Mylopoulos, J.: Non-functional requirements in software engineering. Kluwer, Dordrecht (2000)
4. Dardenne, A., van Lamsweerde, A., Fickas, S.: Goal-directed requirements acquisition. Science of Computer Programming 20(1-2), 3–50 (1993)
5. De Giacomo, G., Lespérance, Y., Levesque, H.: ConGolog, a concurrent programming language based on the Situation Calculus. Artificial Intelligence 121(1-2), 109–169 (2000)
6. Fuxman, A., Liu, L., Mylopoulos, J., Pistore, M., Roveri, M., Traverso, P.: Specifying and analyzing early requirements in Tropos. Requirements Engineering Journal 9(2), 132–150 (2004)
7. Gans, G., Jarke, M., Kethers, S., Lakemeyer, G.: Continuous requirements management for organisation networks: a (Dis)trust-based approach. Requirements Engineering Journal 8(1), 4–22 (2003)
8. Jennings, N.R.: Agent-oriented software engineering. In: Garijo, F.J., Boman, M. (eds.) MAAMAW 1999. LNCS, vol. 1647, pp. 1–7. Springer, Heidelberg (1999)
9. van Lamsweerde, A., Darimont, R., Massonet, P.: Goal-directed elaboration of requirements for a meeting scheduler: problems and lessons learnt. In: Proc. RE 1995, York, UK (1995)
10. van Lamsweerde, A.: Requirements engineering in the year 00: a research perspective. In: Proc. ICSE 2000, Limerick, Ireland (2000)
11. Lapouchnian, A.: Modeling mental states in requirements engineering - an agent-oriented framework based on i* and CASL. M.Sc. Thesis. Department of Computer Science, York University, Toronto (2004)
12. Lapouchnian, A., Lespérance, Y.: Modeling mental states in agent-oriented requirements engineering. In: Dubois, E., Pohl, K. (eds.) CAiSE 2006. LNCS, vol. 4001, pp. 480–494. Springer, Heidelberg (2006)
13. Lespérance, Y., Kelley, T.G., Mylopoulos, J., Yu, E.S.K.: Modeling dynamic domains with conGolog. In: Jarke, M., Oberweis, A. (eds.) CAiSE 1999. LNCS, vol. 1626, pp. 365–380. Springer, Heidelberg (1999)
14. Lespérance, Y.: On the epistemic feasibility of plans in multiagent systems specifications. In: Meyer, J.-J.C., Tambe, M. (eds.) ATAL 2001. LNCS, vol. 2333, pp. 69–85. Springer, Heidelberg (2002)

15. McCarthy, J., Hayes, P.: Some philosophical problems from the standpoint of artificial intelligence. Machine Intelligence 4, 463–502 (1969)
16. Moore, R.C.: A formal theory of knowledge and action. In: Hobbs, J.R., Moore, R.C. (eds.) Formal Theories of the Common Sense World, pp. 319–358. Ablex Publishing, Greenwich (1985)
17. van Otterloo, S., van der Hoek, W., Wooldrige, M.: Model checking a knowledge exchange scenario. Applied Artificial Intelligence 18(9-10), 937–952 (2004)
18. Reiter, R.: The frame problem in the Situation Calculus: a simple solution (sometimes) and a completeness result for goal regression. In: Lifschitz, V. (ed.) Artificial Intelligence and Mathematical Theory of Computation: Papers in Honor of John McCarthy, pp. 359–380. Academic Press, London (1991)
19. Reiter, R.: Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems. MIT Press, Cambridge (2001)
20. Sardina, S., Vassos, S.: The Wumpus world in IndiGolog: A Preliminary Report. In: Proc. Sixth Workshop on Nonmonotonic Reasoning, Action, and Change (NRAC 2005) at IJCAI 2005, Edinburgh, UK (2005)
21. Scherl, R.B., Levesque, H.: Knowledge, action, and the frame problem. Artificial Intelligence 144(1-2), 1–39 (2003)
22. Shapiro, S.: Specifying and Verifying Multiagent Systems Using CASL. Ph.D. Thesis. Dept. of Computer Science, University of Toronto (2004)
23. Shapiro, S.: Modeling Multiagent Systems with CASL - A Feature Interaction Resolution Application. In: Castelfranchi, C., Lespérance, Y. (eds.) ATAL 2000. LNCS, vol. 1986, pp. 244–259. Springer, Heidelberg (2001)
24. Shapiro, S., Lespérance, Y., Levesque, H.: The Cognitive Agents Specification Language and verification environment for multiagent systems. In: Proc. AAMAS 2002, Bologna, Italy, pp. 19–26 (2002)
25. Shapiro, S., Pagnucco, M., Lespérance, Y., Levesque, H.: Iterated belief change in the Situation Calculus. In: Proc. KR-2000, Breckenridge, Colorado, USA (2000)
26. Wang, X.: Agent-oriented requirements engineering using the ConGolog and i* frameworks. M.Sc. Thesis. Department of Computer Science, York University, Toronto (2001)
27. Wang, X., Lespérance, Y.: Agent-oriented requirements engineering using ConGolog and *i**. In: Proc. AOIS 2001 (2001)
28. Wooldridge, M.: Agent-based software engineering. IEE Proceedings on Software Engineering 144(1), 26–37 (1997)
29. Yu, E.: Modeling strategic relationships for process reengineering. Ph.D. Thesis. Department of Computer Science, University of Toronto (1995)
30. Yu, E.: Towards modeling and reasoning support for early requirements engineering. In: Proc. RE 1997, Annapolis, MD, USA (1997)

Author Index