Lecture Notes in Computer Science 1691
Edited by G. Goos, J. Hartmanis and J. van Leeuwen

Springer
Berlin
Heidelberg
New York
Barcelona
Hong Kong
London
Milan
Paris
Singapore
Tokyo

Johann Eder Ivan Rozman
Tatjana Welzer (Eds.)

Advances in Databases and Information Systems

Third East European Conference, ADBIS'99
Maribor, Slovenia, September 13-16, 1999
Proceedings

Springer

Series Editors

Gerhard Goos, Karlsruhe University, Germany
Juris Hartmanis, Cornell University, NY, USA
Jan van Leeuwen, Utrecht University, The Netherlands

Volume Editors

Johann Eder
Faculty of Economics, Business Administration and Informatics
University of Klagenfurt
Uiversitätsstr. 65-67, 9020 Klagenfurt, Austria
E-mail: eder@isys.uni-klu.ac.at

Ivan Rozman
Tatjana Welzer
Faculty of Electrical Engineering and Computer Science
University of Maribor
Smetanova 17, 2000 Maribor, Slovenia
E-mail: {i.rozman/welzer}@uni-mb.si

Cataloging-in-Publication data applied for

Die Deutsche Bibliothek - CIP-Einheitsaufnahme

Advances in databases and information systems : third east European conference ; proceedings / ADBIS '99, Maribor, Slovenia, September 13 - 16, 1999. Johann Eder . . . - Berlin ; Heidelberg ; New York ; Barcelona ; Hong Kong ; London ; Milan ; Paris ; Singapore ; Tokyo : Springer, 1999
(Lecture notes in computer science ; Vol. 1691)
ISBN 3-540-66485-8

CR Subject Classification (1998): H.2, H.5.1, H.4.3, H.3, I.2.4

ISSN 0302-9743
ISBN 3-540-66485-8 Springer-Verlag Berlin Heidelberg New York

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

© Springer-Verlag Berlin Heidelberg 1999
Printed in Germany

Typesetting: Camera-ready by author
SPIN: 10704478 06/3142 – 5 4 3 2 1 0 Printed on acid-free paper

Preface

The 3rd Eastern European Conference on Advances in Databases and Information Systems (ADBIS'99) took place on 13-16- September 1999, in Maribor, Slovenia.

It was organized in cooperation with ACM SIGMOD, the Moscow chapter of ACM SIGMOD, Slovenian Society Informatika, and the Slovenian Ministry of Technology.

The aim of the ADBIS series of conferences is to provide a forum for the exchange of scientific achievements and experiences using innovative methods and approaches between the research communities of Central and Eastern Europe and the rest of the world in the area of databases and information systems.

The 1999 conference continues the series of ADBIS events held in Moscow, St. Petersburg, and Poznan. The ADBIS steering committee has the ambition to make the ADBIS conference the premiere database and information systems conference in Central and Eastern Europe, to increase interaction and collaboration between researchers from East and West, and to provide an internationally recognized forum for the presentation of research and experiences in all aspects of modern database technology and information systems.

To achieve these goals an international program committee selected 26 full research papers in a rigorous reviewing process from a total of 94 submissions from 33 different countries. This high number of submissions from so many different areas shows the truly worldwide recognition of and interest in the ADBIS series.

The accepted papers cover a wide range of topics from storage structures to web applications, from type theory to performance issues. Furthermore, three invited papers give a broader overview of research achievements and insights into the current "hot topics" of databases and information systems: Internet and multimedia, electronic commerce, and ODMG object models.

In addition to the presentation of invited papers and full research papers, the program consists of three tutorials and the presentation of short papers that either allow a glimpse into promising ongoing projects, present novel ideas at an early stage, or give an account of experiences with the application of databases and information systems. These short papers are published in a separate proceedings volume.

We would like to express our thanks and acknowledgements to all the people who have contributed to make ADBIS'99 a success:

- All the organizers of the previous ADBIS workshops and conferences. They made ADBIS a valuable trademark and we are proud to continue their work.
- The authors, who submitted papers of high quality to the conference.
- ACM SIGMOD and the international program committee for ensuring the high quality of the scientific program.
- Ludvik Toplak and the University of Maribor for supporting ADBIS'99 activities.
- Bruno Stiglic for leading the organizing committee.
- Bostjan Brumen to whom we would like to express our special thanks for his continuous and coordinating activities that ensured the success of the ADBIS'99.
- Izidor Golob for setting up and maintaining the ADBIS'99 home page and assembling the proceedings together with Chrystian Barragan.

- Springer-Verlag for publishing these proceedings and Alfred Hofmann for the effective support in producing these proceedings.
- And last but not least we thank the steering committee and, in particular, its chairman, Leonid Kalinichenko, for its advice and guidance.

June 1999

Johann Eder
Ivan Rozman
Tatjana Welzer

Conference Organization

General Chair

Ivan Rozman (University of Maribor, Slovenia)

Honorary Chair

Ludvik Toplak (Rector of University of Maribor, Slovenia)

ACM SIGMOD Advisor

Marek Rusinkiewicz (MCC, USA)

European Coordinator

Rainer Manthey (University of Bonn, Germany)

Program Committee Co-Chairs

Tatjana Welzer (University of Maribor, Slovenia)
Johann Eder (University of Klagenfurt, Austria)

Program Committee

Divyakant Agrawal (UC Santa Barbara)
Suad Alagic (Wichita State University)
Yuri Breitbart (Bell Laboratories)
Jerzy Brzezinski (Poznan University of Technology)
Omran Bukhres (Purdue University)
Wojciech Cellary (University of Economics at Poznan)
Jan Chomicki (Monmouth University)
Bogdan Czejdo (Loyola University)
Michael Dobrovnik (University of Klagenfurt)
Nikolay Emelyanov (Russian Academy of Sciences)
Matjaz Gams (Institute Josef Stefan)
Janis Grundspenkis (Riga Technical University)
Remigijus Gustas (University of Karlstad)

Jozsef Gyorkos (University of Maribor)
Abdelsalam Helal (University of Florida)
Tomas Hruska (Technical University Brno)
Hannu Jaakkola (Tampere University of Technology [Pori])
Leonid Kalinichenko (Russian Academy of Sciences)
Wolfgang Klas (University of Ulm)
Mikhail Kogalovsky (Russian Academy of Sciences)
Peter Kokol (University of Maribor)
Ralf Kramer (Forschungszentrum Informatik [FZI])
Sergey Kuznetsov (Russian Academy of Sciences)
Pericles Loucopoulos (UMIST)
Ronald Maier (University of Georgia)
Florian Matthes (TU Hamburg-Harburg)
Yannis Manolopoulos (Aristotle University)
Rainer Manthey (Bonn University)
Guido Moerkotte (University of Aachen)
Tomaz Mohoric (University of Ljubljana)
Tadeusz Morzy (Poznan University of Technology)
Pavol Navrat (Slovak University of Technology)
Boris Novikov (University of St. Petersburg)
Alain Pirotte (University of Louvain)
Misha Rabinovich (AT&T)
Tore Risch (Linkoping University)
Colette Rolland (University of Paris 1 - Sorbonne)
Silvio Salza (University of Rome “La Sapienza”)
Michel Scholl (CNAM and INRIA)
Julius Stuller (Czech Academy of Science)
Kazimierz Subieta (Polish Academy of Science)
Bernhard Thalheim (TU Cottbus)
Bostjan Vilfan (University of Ljubljana)
Gottfried Vossen (University of Muenster)
Benkt Wangler (Stockholm University)
Tatjana Welzer (University of Maribor)
Viacheslav Wolfengagen (Institute for Contemporary Education "JurInfoR-MSU")
Vladimir Zadorozhny (University of Maryland)
Alexander Zamulin (Institute of Informatics Systems)
Damjan Zazula (University of Maribor)

Organizing Committee

Chair:
Bruno Stiglic (University of Maribor, Slovenia)

Members:
Bostjan Brumen
Izidor Golob
Chrystian Barragan
Matjaz B. Juric
Tomaz Domajnko

Miha Strehar
Stefan Masic
Tomislav Rozman
Simon Beloglavec
Boris Lahovnik
Robert T. Leskovar
Ales Zivkovic
Marjan Hericko
Romana Vajde Horvat

ADBIS Organization

Steering Committee Chair

Leonid Kalinichenko (Russian Academy of Sciences, Russia)

Steering Committee Members

Radu Bercaru (Romania)
Albertas Caplinskas (Lithuania)
Janis Eiduks (Latvia)
Hele-Mai Haav (Estonia)
Mikhail Kogalovsky (Russia ACM)
Tadeusz Morzy (Poland)
Pavol Navrat (Slovakia)
Boris Novikov (Russia ACM)
Jaroslav Pokorny (Czech Republic)
Anatoly Stogny (Ukraine)
Tatjana Welzer (Slovenia)
Viacheslav Wolfengagen (Russia ACM)

Additional Referees

Alex Nanopoulos
Andrej Sostaric
Apostolos Papadopoulos
Bernhard Thalheim
Bostjan Brumen
Cezary Sobaniec
Viljan Mahnic
Daniel Lieuwen
Danilo Korze
Danny Brash
Dariusz Wawrzyniak
Dean Korosec
E. Bertino
Ekaterina Pavlova
Elke Hochmüller
Eriks Sneiders

Giovanni Rumolo
Giuseppe Santucci
Heinz Frank
Henrik Andre-Jönsson
Hua Shu
Igor Nekrestyanov
Janis Stirna
Jerzy Stefanowski
Klaus-Dieter Schewe
Maciej Zakrzewicz
Marjan Druzovec
Marlon Dumas
Matjaz Branko Juric
Michal Szychowiak
Georgios Giannopoulos
Nils Klarlund
Nur Yilmazturk
Paolo Merialdo
Rafal Chmielewski
Rainer Schmidt
Robert Wrembel
Robin Lindley
S. Seshadri
Sara Holmin
Srinathh Srinivasa
Stanislaw Ambroszkiewicz
Svetlana Kouznetsova
Thomas Feyer
Tomaz Domajnko
Vili Podgorelec
Xun Cheng

Table of Contents

Invited Papers

Trust for Electronic Commerce Transactions..... 1
Günther Pernul, Alexander W. Röhm, Gaby Herrmann

A Family of the ODMG Object Models 14
Suad Alagić

Regular Papers

Integration

Consistent Handling of Integrity Constraints and Extensional Assertions for Schema Integration.....31
Can Türker, Gunter Saake

Deriving Type Conflicts and Object Cluster Similarities in Database Schemes by an Automatic and Semantic Approach.....46
Domenico Ursino

Information Retrieval & Multimedia

A Model for Querying Annotated Documents 61
Frédérique Laforest, Anne Tchounikine

Word-Based Compression Methods and Indexing for Text Retrieval Systems.....75
Jiri Dvorský, Jaroslav Pokorný, Vaclav Snášel

Processing of Spatiotemporal Queries in Image Databases 85
Theodoros Tzouramanis, Michael Vassilakopoulos, Yannis Manolopoulos

Transactions

On the Correctness of Virtual Partition Algorithm in a Nested Transaction Environment 98
Sanjay Kumar Madria, S.N. Maheshwari, B. Chandra

Computing Rules for Detecting Contradictory Transaction Termination Dependencies.....113
Kerstin Schwarz, Can Türker, Gunter Saake

A New Class Hierarchy Concurrency Control Technique in Object-Oriented Database Systems.....128
Woochun Jun

Index-Structures

The G^r_Tree: The Use of Active Regions in G-Trees 141
Dimitris G. Kapopoulos, Michael Hatzopoulos

S*-Tree: An Improved S^+-Tree for Coloured Images 156
Enrico Nardelli, Guido Proietti

Agent Technology

Workflow Management System Using Mobile Agents 168
Zoran Budimac, Mirjana Ivanović, Aleksandar Popović

Data Mining

Pattern-Oriented Hierarchical Clustering 179
Tadeusz Morzy, Marek Wojciechowski, Maciej Zakrzewicz

Data Mining in a Multi-dimensional Environment 191
Holger Günzel, Jens Albrecht, Wolfgang Lehner

Arbiter Meta-learning with Dynamic Selection of Classifiers and Its Experimental Investigation 205
Alexey Tsymbal, Seppo Puuronen, Vagan Terziyan

Data Modeling

The Notion of ´Classes of a Path´ in ER Schemas 218
Junkang Feng, Malcolm Crowe

Evaluation of Data Modeling 232
Ronald Maier

Organizational Modeling for Efficient Specification of Information Security Requirements 247
Jussipekka Leiwo, Chandana Gamage, Yuliang Zheng

Incomplete Data

A Closed Approach to Vague Collections in Partly Inaccessible Distributed Databases 261
Oliver Haase, Andreas Henrich

Infinite Relations in Paraconsistent Databases 275
Nicholas Tran, Rajiv Bagai

Queries

Query Rewriting and Search in CROQUE 288
Joachim Kröger, Regina Illner, Steffen Rost, Andreas Heuer

Object Query Optimization in the Stack-Based Approach 303
Jacek Płodzień, Anna Kraken

Database Theory

Compositional Specification Calculus for Information Systems Development 317
Leonid Kalinichenko

Performance Issues

The Impact of Using Class Inheritance in a Distributed Information System 332
Patrick Dunne, Alex Gray

Performance Assessment Framework for Distributed Object Architectures 349
Matjaž B. Jurič, Tatjana Welzer, Ivan Rozman, Marjan Heričko, Boštjan Brumen, Tomaž Domajnko, Aleš Živkovič

Efficient Use of Signatures in Object-Oriented Database Systems 367
Kjetil Nørvåg

Author Index 383

Trust for Electronic Commerce Transactions

Günther Pernul, Alexander W. Röhm, and Gaby Herrmann

Department of Information Systems
University of Essen, Germany
{pernul,roehm,herrmann}@wi-inf.uni-essen.de

Abstract. The dramatic changes in telecommunications and computing technology as evidenced in the Internet and WWW have sparked a revolution in electronic commerce (e-commerce). In fact, many organisations are exploiting the opportunities of Internet-based e-commerce solutions, and many more are expected to follow. But in spite of the well-published success stories, many businesses and consumers are cautious about e-commerce, and security concerns are often cited as being the most important barrier. In this paper, we identify security and fairness in e-commerce transactions as basic requirements demanded by any participant in electronic markets. We discuss different phases of e-commerce transactions and study security requirements which are important to guarantee during each of the phases. In order to develop trust for e-commerce transactions we propose 1. COPS, a technical infrastructure for building adaptable electronic markets with main focus on security and fairness, and 2. MOSS, a business process reengineering methodology for analysing and modelling the security semantics of business transactions in order to transfer them to electronic markets. Both, COPS and MOSS are helpful to control the risks involved in dealing (trading) with untrusted parties in an open e-commerce environment.

1 Introduction

Obviously there are lots of advantages in performing business electronically via telecommunication networks. It is therefore not surprising that many organisations are exploiting the opportunities offered by Internet-based e-commerce solutions, and that many more are expected to follow. There are already some well-published success stories of companies and organisations being active in e-commerce; however, many businesses and consumers are also still cautious about participating in e-commerce, and security concerns are often cited as being the single most important barrier.

There are some very fundamental requirements which inhibit, or at least slow down the success and growth of e-commerce. The most important requirements, among others, are the lack of

- a generic and user-friendly infrastructure supporting e-commerce,
- security and fairness as integral parts of the infrastructure,
- methodologies for analysing and modelling the security semantics of e-commerce business transactions.

Eder et al. (Eds.): ADBIS´99, LNCS 1691, pp.1 -13, 1999.
© Springer-Verlag Berlin Heidelberg 1999

In order to justify our statement consider as an example an electronic market[1]. The *generic and user friendly infrastructure* is mainly necessary for the demander. The demander usually is the driving force in a business transaction and thus the infrastructure must enable him easy access to an electronic market. Otherwise, many potential demanders will not participate in the electronic market which could endanger all the benefits of e-commerce.

The requirement *security and fairness* is necessary for all market participants. A supplier will not offer goods if, for example, secure payment is not guaranteed. The same is true for a demander. He will not participate if, for example, delivery is not guaranteed after payment. Both may not participate in the market if they are not treated by the market infrastructure in a fair way and if privacy and confidentiality is not guaranteed in a business transaction.

In order to develop trust in e-commerce transactions and supporting infrastructures it is additionally essential to know about all security risks of a business transaction. This requires a careful analysis of all security relevant knowledge involved in processing the transaction. We call this knowledge the *security semantics* of the business transaction.

While the first two requirements are enabling services for e-commerce the third requirement is necessary because on our way into the information age we are currently in a phase of transformation. E-commerce forces rapid changes in the business behaviour of trading partners. The changes will lead to a reorganisation of well-established procedures and thus may make corrections of existing systems necessary which – as a side effect – may also have consequences to their security.

In our current research we deal with all these three issues: We are developing a technological infrastructure for establishing an electronic market as a basis for secure and fair e-commerce. The infrastructure is called COPS (commercial protocols and services) and its main emphasis is to develop trust for e-commerce transactions by focusing on security and fairness requirements of market participants in each of the phases of a business transaction. It is generic because of two reasons: First, it is extensible in the sense that whenever new requirements or security threads occur they may be easily integrated into COPS. Second, it is adaptable in the sense that it supports different market structures. This may be necessary because different goods may be traded on different types of markets (i.e. direct search markets, brokered markets or auction markets). In a second project, it is called $MOSS_{BP}$ (modelling security semantics of business processes) we are developing a business process re-engineering methodology in order to analyse the security semantics of business processes. The re-engineering aspect is of particular importance in order to transfer a traditional business transaction into an equivalent e-commerce business transaction.

[1] Similar to a conventional market an electronic market is a mechanism for coordinating the allocation of goods (between supplier and demander) but instead of acting on the market in a traditional (physical presence, paper-based) way its participants extensively make use of information technology.

The goal of this paper is to give a general overview of our current research in developing trust for e-commerce business transactions. It is structured as follows: In section 2 we give basic definitions and examples of security and fairness requirements. In section 3 we introduce the COPS project. In section 4 we discuss the ideas behind $MOSS_{BP}$. Section 5 describes the relationship between COPS and MOSS and concludes the paper.

2 Security and E-commerce

In this section we will first define the term security, than discuss electronic business transactions, electronic markets and finally give examples of security requirements in the different phases of an e-commerce business transaction.

2.1 Basic Definitions

Some words on the term “security” are in order. There is no general agreement about the term and its semantics may also have changed over time. This paper uses a very broad definition of security for e-commerce. We subsume under *security* (non exhaustive list only):

- the general definition: confidentiality, integrity, availability
- intellectual property involved in digital goods: authorship, ownership, copyright, right to use, originality,
- bindings: legal binding, mutual dependencies, non-repudiation
- privacy, anonymity, fairness, trust.

A well accepted definition of *e-commerce* is that it "... is the sharing of business information, maintaining business relationships and conducting business transactions by the means of telecommunication networks." [1]. An important part of e-commerce according to this definition are business transactions, that may take place within organisations, between organisations or in an electronic market in order to allocate goods between different market participants.

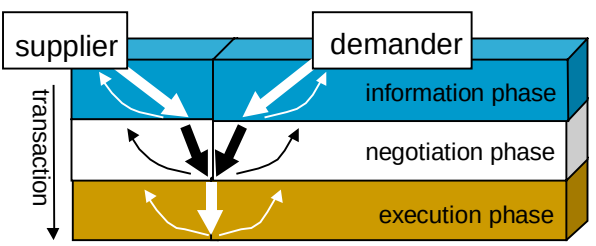


Fig. 1. Business transaction phases

Any business transaction includes at least three phases (see fig. 1): In the *information phase* the demander (in few cases also the supplier) is searching for a matching business partner. For this purpose a demander asks potential suppliers for their offers. After he has collected these offers he chooses the one he prefers. After matching business partners are found the information phase terminates. During the *negotiation phase* the supplier and the demander have to find an agreement. Many details of the contract have to be fixed like the method of payment, the method of shipping and many others. All obligations for the supplier as well as the demander have to be mentioned in this contract. In the case all contract partners agree the negotiation phase terminates. During the *execution phase*, both the demander and the supplier have to meet the obligations described and fixed in the contract that they have made in the negotiation phase. The execution phase usually consists of two sub-phases: the payment and the delivery of goods phases. Both strongly depend on the type of the good and on the type of the market where the good is offered.

The different phases of a business transaction are processed in an interactive and iterative way. For example, if two matching partners cannot agree about signing a contract the negotiation phase terminates and the information process starts again. A market in which the three phases are supported by information and communication technology is called an *electronic market*. There also exists a broader definition which says that at least one of the three phases has to be supported by information and communication technology for a market to be an electronic market.

Each business transaction on an electronic market involves special risks that depend on the type of (electronic) media, the good and its value, the type of market and of course on the type of attacks that might occur. As a consequence for each phase of a market transaction certain security considerations are necessary. In an infrastructure for a secure electronic markets we have to cover a whole variety of security risks in all three phases by offering appropriate security services.

2.2 Examples of Security Requirements

During the *information phase* the market participants do have different security demands. When browsing through the Internet a demander has to be sure that an offer he is considering is still valid. Additionally, he must even be sure that the supplier is still in operation under the network address he found the offer. In the case both pre-conditions are valid and in order to reduce the trading risk the demander wants to accept only authenticated offers. But there are also security requirements for the supplier: For example, the supplier may want his offers to be confidential because otherwise the competitors may gain business advantages.

The need for security services in the *negotiation phase* is obvious. Most important is the legal binding of contract partners. The contract and its security services must include enough information to determine who is right when demander and supplier later disagree. During this phase the probably most important security demands are integrity, authenticity and the legal binding of the contract partners on the content of the contract.

In the case that the *execution phase* is conducted electronically both the electronic payment and the transfer of goods have to be secure. For secure payments several proposals have been made by academics as well as industries. They offer different levels of security and privacy. Secure delivery of digital goods can have many different security demands depending on the type of good that has to be exchanged. For example digital represented shares have to be original but an additional (and often conflicting) requirement is that they have to be anonymous (they do not contain the name or even a pseudonym of their owner) for economic reasons. But there are also security demands in the execution phase that do neither depend on the good nor on the method of payment. For example the fairness problem is evident and not solved by simply making a contract. As an example consider a protocol which assumes delivery of the goods after payment. This is an unfair situation for the demander. The same is true for the supplier if delivery proceeds payment. Additionally the supplier needs a confirmation when he delivers the goods in order to prove that he has met his obligations. The same problem exists with the demander vice versa.

The security of e-commerce is influenced by the different phases of a business transaction, the market participants, the type and value of the digital good, the type of the market (market structure). Because of the diversity of the different security requirements it is necessary to have a clear understanding about all these key factors influencing security and fairness in e-commerce.

3 COPS - An Infrastructure for Secure and Fair E-commerce

In this section we will discuss the COPS infrastructure model, the involved market participants in a COPS electronic market, and the software architecture of COPS. Additional information about COPS may be found in [2], [3], and [4].

3.1 COPS – Infrastructure Model

COPS has the goal to develop and implement a prototype of a generic market infrastructure for e-commerce with its main focus on security and fairness for market participants. Each market participant in each phase of the business transaction is defined (1) by a set of services offered by the infrastructure, (2) by its communication relationships to other market participants, and (3) by a set of actions, conditions and events which are characteristic for this part of the business transaction.

In figure 2 we give a graphical model of the underlying market infrastructure of COPS. The three levels (I,N,E) show the three phases of the business transactions, while the corner elements are representing the different participants in a COPS market.

Market-based co-ordination can be classified into four categories: direct-search markets (where the future partners directly seek out one another), brokered markets (with the brokers assuming the search function), dealer markets (with the dealers holding inventories against which they buy and sell), and the auction markets

[1], [5]. From this classification we derive four electronic market player roles: demander, supplier, electronic intermediary (cybermediary) and trusted third party. Together with the information services five roles of participants are considered in the COPS market model:

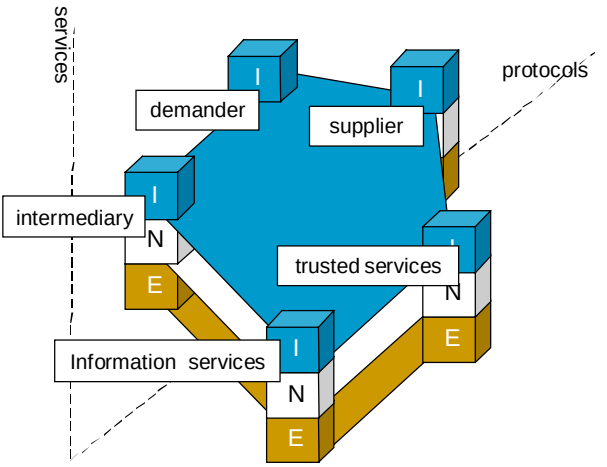


Fig. 2. COPS Market Model

- The *demander* is the driving force of a transaction. Only in the information phase it is possible that a supplier offers products. All other phases are initiated by the demander. It is on the demander's side, where an infrastructure has to preserve the open character of an electronic market. In particular no or less technical or organisational preconditions for a demander to participate should exist. This openness generally includes, that there is no trust relationship between the business partners, which may lead to additional security threads [6].
- The *supplier* has the choice to offer her/his goods either on a direct search market, through cybermediaries or on an electronic auction market. The choice will depend on the suppliers preferences, on the type of the good offered and on other strategic considerations.
- An electronic *intermediary* is trading information about products like their prices or quality. He offers product evaluation, quality assurance, or special combinations of products (e.g. travel agency). There are quite different understandings of electronic intermediaries. All agree, that intermediaries will survive (despite the fact that direct producer buyer relationships are easier) in electronic markets, because they are able to produce an added value to electronic goods [7].
- *Trusted third parties* play an important role in security infrastructures, because they are used for public-key certification. Public-key certification is a necessary requirement to support, for example, authenticity of market participates which may be necessary for legal binding of contract partners on the contract. This is practically important for contracting but also for digital goods which often need authenticity, originality and similar properties. For future electronic markets we expect a lot of new

tasks for trusted third parties, for example, auction services, notary services, timestamp services, registration services.

- *Information services* provide technical information about market infrastructure and network. Examples are certificate directories or a hosts which processes inquiries like: "What is the network address of a trusted third party issuing secure time stamps?".

An application of COPS is given in [8]. It shows the completion phase of an electronic market for free trading of anonymous and original emission permits.

3.2 COPS Prototype – Software Architecture

The architecture of the COPS prototype is shown in figure 3. COPS is based on a layered architecture. At the lowest level it uses Internet transport protocols for communication and database functionality to store, manage and retrieve data.

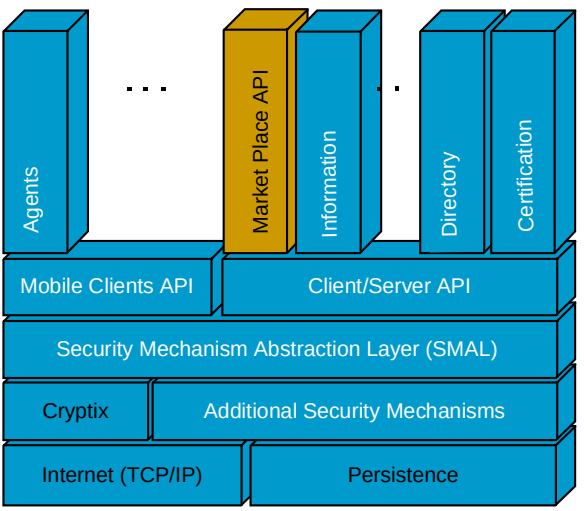


Fig. 3. COPS-Prototype- Software Architecture

The two layers in the middle are included to support abstraction levels, that allow to integrate other security solutions like SSL at the communications layer or new cryptographic systems at the security mechanism layer. Currently, the second layer includes a commercial security library extended by some additional security mechanisms. The SMAL (Security Mechanism Abstraction Layer) offers security services in a transparent way and allows to use cryptographic mechanisms from the second layer without the need to know about their implementation. This makes it easier to change the functionality of COPS without changing other parts of the software and makes COPS flexible if new standards come up or stronger cryptographic mechanisms are found. On top of the SMAL is the communication layer on which the application interfaces for the service modules of COPS are located. Presently we are working on two different libraries, that are part of this

layer: The CsAPI Client/Server API, which provides secure interfaces for market communication to the upper layers and the McAPI (Mobile Client API) for agent communication. Since the McAPI is considered to be used for mobile agents it will be as small as possible for minimal overhead when agents are moving through the net.

On top of the application interfaces McAPI and CsAPI basic service modules are located. Some possible service modules are the information services, public-key certificate directories, or certification authorities. At the same layer a further application interface is located: the Market Place API. Part of this interface are the implementations of the services, that are specific for a specific role in a certain type of market.

4 MOSS – An Environment for BPR

In order to develop trust in e-commerce transactions and supporting infrastructures it is essential to know about all security risks which may be involved in the business processes responsible for carrying out the e-commerce transaction. This requires a careful analysis of all security relevant knowledge involved in processing the transaction. We call this knowledge the *security semantics* of the business transaction. In the project $MOSS_{BP}$ (modelling security semantics of business processes) we are developing a business process re-engineering methodology in order to analyse the security semantics of business processes. The re-engineering aspect is of particular importance if one wishes to transfer a traditional business transaction into an equivalent e-commerce business transaction. In this section we give an overview of MOSS. Additional information may be found in [9] and [10].

4.1 MOSS – Overall Architecture

It is a long way from specifying and modelling requirements to their final processing. For enforcing security semantics we propose the following framework: A domain expert analyses and specifies business processes of the application domain including their security and integrity requirements at a high level of abstraction. The executive responsible for specifying the business process usually is not a security specialist. At a very high level only he has knowledge about requirements like “sensitivity”, “legal binding”, “high integrity”, “copyright”, and similar, and will assign the requirements to business process components. The security requirements will later on be analysed by an security expert. He will be responsible for analysing the requirements and together with the domain expert for embedding the requirements into the e-commerce transactions.

MOSS consists of a three-layered architecture (see fig. 4**).** The top layer (third layer) is divided into two sublayers. The sublayer 3.1 contains well-defined concepts used to represent the security semantics of the real world and a method to analyse them from different perspectives. The domain expert may use these concepts to

develop a business process model of the e-commerce transaction and to express its security requirements. The sublayer 3.2 contains a repository with reusable case studies for handling security requirements. These case studies look at security requirements in more detail and offer possible ways for their realisation.

	description of content	**representation method**	**supporting method**
3	high-level and abstract specifications	Graphical model	• business process model security semantics • repository of case studies
2	detailed specifications (security semantics only)	intermediate language ALMO$T	transformation rules from layer 3 to layer 2
1	security hardware and software building blocks	programs, hardware	crypto-library, security APIs, security dongle, ...

Fig. 4. MOSS - Three-layered architecture

To transform the specified security requirements into an executable form additional work is necessary. For that, detailed knowledge of security concepts is required. The security administrator takes the high-level security specifications as input and transforms them in a more detailed representation. This may also be done by consulting the repository from sublayer 3.2. Layer 2 of the architecture contains guidelines about dividing security activities into basic building blocks (expressed in an intermediate language). Layer 1 of the architecture offers a repository of hardware and software components which may be needed to realise the building blocks. In the following section we will comment on layer 3 of the architecture only.

4.2 MOSS – Perspectives and Views

In general, a business process is described by a process model which contains information about the process characteristics relevant to the purpose of the business target. According to [11] a combination of the following four perspectives produces an integrated, consistent, and complete view of a business process:

- The *informational perspective* represents the information entities, their structuring and relationships between them.
- The *functional perspective* shows what activities (processes) are performed and which data flow occurs between activities. The functional perspective only represents the flow of data within the system.
- The *dynamical perspective* represents for each information entity all possible states and state transitions which may occur within the life cycle of the information entity.

- The *organisational perspective* shows where and by whom activities are performed. This perspective corresponds to the organigram of an organisation and to role models.

Each perspective focuses on a very specific part of a business process and to specify and to understand the perspectives some knowledge about modelling techniques is necessary. Usually, for the expert in the business domain who is specifying the business process it is very difficult to see the relationships between the business process and other models already existing in the organisation. For analysing security semantics it is very important to develop an integrated view of all aspects of a business process. In addition to the four perspectives already mentioned the MOSS framework supports a fifth perspective. We call this additional perspective the business process perspective.

- The *business process perspective* represents the flow of work in terms of activities and information flow from the viewpoint of the whole business process. It consists of an integration of the functional and dynamic perspectives and references the informational and organisational perspectives.

4.3 MOSS – Example "Contract Signing"

A business process is integrated in the totality of activities of the enterprise. Before a new business process will be initiated usually the enterprise and other business processes already exist and at least parts of them are already modelled. Therefore, models may exist of the organisational structure of the enterprise (organisational perspective), of the structure of a database (informational perspective), of already existing activities (processes) and data flow between them (functional perspective), and of the life cycles of the information entities (dynamic perspective). The model of a new business process must relate to these existing models and must extend them accordingly. Thus, some sort of re-engineering may be appropriate.

In this subsection we focus on the informational perspectives of contract completion and the security requirement "legal binding" of the contract partners. In the example we use the following notation: Components of existing models which are not effected by security requirements are written using standard characters. The attributes with relevance to legal binding are given in bold face.

To guarantee legal binding of a contract different regulations are required according to corresponding law. In many countries a contract is legally binding if the contract partner agree about its content. For the case the contract partner later on disagree the agreement must be provable. For provability a common method is the use of signatures. If the contract is a traditional document (by paper) a way to realise legal binding of the contract is to have it signed by the contract partners. In our example we examine the situation in which the document is part of an e-commerce transaction and available in digital form only. The example is based on the IuKDG [12] in Germany, a bill for digital signatures. This bill requests the following for legal binding of an digital signature: A digital signature is a seal based on the signed

data. The seal is realised by asymmetric cryptography and is created by using the private key of the signatory. It is possible to ascertain the correctness of signed data and the identity of the signatory by using its public key which must be certified by a certification authority. Each certificate is assigned with a period of validity.

In order to study what effects a digital signature has, we will refer to the informational perspective of a business process, for example “order management”. As contracting information is usually stored in a database we assume the existence of a data model at the demander side covering the demander-supplier relationship. In order to process the business transaction on an electronic market and to establish legal binding, the informational perspective of the business process must be extended by information about the signatories, the certificates used, and the (trusted) parties responsible for issuing the certificates. In figure 5 we have extended the customer-supplier-relationship by appropriate data structures necessary for supporting legal binding. In particular these are: a new relationship type `certificate` and modification of the existing relationship type representing the `contract`. The agreed fact is represented by a document which should be signed and the relationship type between `customer` and `supplier` must be extended by one field for the seal (digital signature) of each signatory and by information about what algorithm was used for signing. In addition, customer and supplier are specialisations of a generic type `signatory` which must have a `certificate` relating the applicant to a `certification authority`.

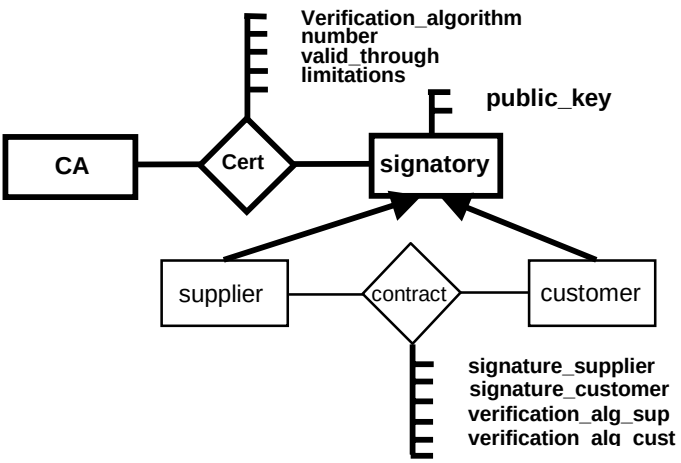


Fig. 4. MOSS informational perspective – legal binding

This example has shown that if one wishes to transfer a “traditional” business transaction into an e-commerce transaction some sort of re-engineering of the enterprise-wide data model (informational perspective) is necessary in order to support the security requirement “legal binding”. To guarantee legal binding of the contract the functional perspective of the business process must be modified as well. The document must be signed digitally by each contract partner and the signatures must be

verified. Because a certificate of a public key may be expired (by time or by declaration of invalidity) additional actions are necessary to guarantee the provability of digital signed contracts. These actions lead e. g. to extensions of the functional perspective of a process responsible for archiving contracts. An e-commerce transaction demanding legal binding has impacts to the organisational perspective, too. To check the validity of digital signatures and to initiate further actions, a new role in the organisation is necessary and leads to an extension of the organisational perspective. Of course, it has also effects on the dynamic perspective of the business process. We might have the state "~valid" of an object of type "contract". It represents a contract which is valid (i.e. signed) but the certificate of the signature is expired. This means we have the situation that a contract is valid but no longer provable. Each contract in this state must be re-signed.

Unfortunately, security requirements of a business transaction have effects on different perspectives of a business process. The example showed that legal binding as needed on electronic markets influences all perspectives of the business process. In MOSS we propose extensions to existing models which seem to be quite complex for the non security expert. However, the outlined extensions are identical for legal binding of a document in any business process requiring this functionality in identical legal environments. Therefore, these extensions may be reused (see sublayer 3.2 of the MOSS architecture).

5 Conclusion

Building trust for e-commerce transactions in open electronic commerce requires the combination of several elements to address commercial, legal, and security concerns. In our ongoing research we are investigating this issues by developing COPS, an electronic commerce infrastructure, and MOSS, a modelling environment for analysing and enforcing business process security. In this paper we have given a short overview of both projects. Additional information may be found in the cited literature.

COPS has its main focus on security and fairness, includes five different roles of market participants, supports all phases of a market transaction, and can be used to build markets with different structures. MOSS can be used to analyse the security semantics of business transactions. It supports different views on a business transaction in order to arrive at a comprehensive picture of the security requirement. Both projects are currently implemented as academic prototypes.

References

[1] Zwass, V.: Electronic Commerce: Structures and Issues. International Journal of Electronic Commerce; Vol. 1; No. 1; M.E. Sharp; 1996.

[2] Pernul, G., Röhm A. W. Integrating Security and Fairness into Electronic Markets. Proc. 4th Research Symposium on Emerging Electronic Markets. Maastricht, September 1997.

[3] Röhm, A. W., Pernul, G.: COPS: A Model and Infrastructure for Secure and Fair Electronic Markets. Proc. of Thirty-Second Hawaii International Conference on System Sciences HICSS-32; (1999).
[4] Röhm, A. W., Pernul, G., Herrmann, G.: Modelling Secure and Fair Electronic Commerce. Proc 14th Annual Computer Security Applications Conference, Phönix, AX, Dec. 1998.
[5] Clemons, E. K.; Croson, D.C.; and Weber, B.W.: Reengineering money: the Mondex stored value card and beyond. International Journal of Electronic Commerce (1997). http://www.cba.bgsu.edu/ijec/
[6] Bons, R. W. H.: Designing Trustworthy Trade Procedures for Open Electronic Commerce. PhD-Series in General Management 27; Rotterdam School of Management; (1997).
[7] Sarkar, M. B.; Butler, B.; and Steinfield, C.: Intermediaries and cybermediaries: a continuing role for mediating players in the electronic marketplace. Journal of Computer-Mediated Communication, Vol. 1, Nr. 3 (1995).
[8] Röhm, A. W., Gerhard, M.: A secure electronic market for anonymous transferable emission permits. Proc. of Thirty-First Hawaii International Conference on System Sciences HICSS-31; (1998).
[9] Herrmann, G., Pernul, G.: Viewing business process security from different perspectives. International Journal of Electronic Commerce; Vol. 3; No. 3; M.E. Sharp; 1999.
[10] Herrmann, G., Pernul, G.: Zur Bedeutung von Sicherheit in interorganisationellen Workflows. WIRTSCHAFTSINFORMATIK, Heft 3, Juni 1997, (in German).
[11] Curtis, B.; Kellner, M.; Over, J.: Process Modeling. Comm. of the ACM, Vol. 35, No. 9, 1992.
[12] Informations- und Kommunikationsdienste-Gesetz – IuKDG, July 1997, (in German).

A Family of the ODMG Object Models*

Suad Alagić

Department of Computer Science,
Wichita State University,
Wichita, KS 67260-0083, USA
alagic@cs.twsu.edu

Abstract. A family of the ODMG Object Models is proposed as a solution to the problems of the ODMG Standard related to the type systems, the model of persistence, and the model semantics. This family is intended to replace the existing single and inadequate ODMG Object Model. Unlike the ODMG Standard, all the models in this family are equipped with orthogonal persistence. The most basic model in the family corresponds to the Java^{TM} programming language. A strictly more sophisticated model features the form of parametric polymorphism suitable for object-oriented database technology. The next model in the hierarchy is equipped with type-safe reflective capabilities. The most sophisticated model is enriched with a logic-based constraint language that is completely missing from the ODMG Standard. Each model in the family is well-defined and avoids the inconsistencies of the ODMG Object Model.

1 Introduction

The ODMG Standard [10] is the only existing proposed standard for object-oriented databases. The Standard has been developed by the major vendors of object-oriented database management systems. The ODMG Object Model is an object-oriented database model intended to serve as the basis for the technology proposed by the ODMG Standard.

The ODMG Standard covers a very complex technology, which includes a variety of languages. This variety includes declarative database languages such as ODL (Object Definition Language) and OQL (Object Query Language). In addition, language bindings are defined for major object-oriented programming languages. At this point, the ODMG Standard specifies bindings for C++, Smalltalk, and Java.

The languages covered by the ODMG Standard are quite different. ODL and OQL were designed separately, and in fact do not match. The same is, of course, true for C++, Smalltalk, and Java. Yet, the ODMG Standard promotes a perfectly valid idea of a unified type system across all these languages. The

* This material is based upon work supported by the NSF under grant number IIS-9811452.

Eder et al. (Eds.): ADBIS'99, LNCS 1691, pp. 14–30, 1999.
© Springer-Verlag Berlin Heidelberg 1999

same should presumably apply to the model of transparent persistence promoted by the Standard.

It is obvious that defining an object-oriented database model that would be common for all these languages presents a major challenge. In fact, we have established that the ODMG Object Model is inadequate for this purpose [1]. The languages for which the ODMG Object Model serves as the common basis are, in fact, incompatible. This is why the model has a number of controversial features [1].

In this paper we address the issue of providing a common basis for the technology proposed by the ODMG Standard. We show that a single object model cannot serve the purpose. We propose a family of models of the increasing level of sophistication. Unlike the existing ODMG Object Model, each model in this family is well-defined, without inconsistencies found in the ODMG Object Model. We also establish the relative advantages and disadvantages of the models in this family.

The strategy that we propose to the vendors is to start supporting the basic model in this family first. There is a convincing argument for this strategy: the basic model corresponds to the Java programming language. Moving to more sophisticated models could be done over time. At each point this strategy makes clear what level of compliance a particular vendor is providing. This eliminates the confusion that exists at present with respect to the level of compliance with the ODMG Standard.

The presentation in the paper is based entirely on the notation and features of the Java programming language. There are, of course, very good reasons for this. The extensions with respect to the current Java technology discussed in the paper are orthogonal persistence, parametric polymorphism, and constraints (assertions). These extensions are in fact requirements for making Java a full-fledged database programming language. In addition, the reflective features in the ODMG Standard (and thus also in the reflective model of the family of the ODMG Object Models) require standard support for dynamic compilation.

The paper is organized as follows. In section 2 we present the most basic model in the family. We also analyze the limitations of this model. All the models in this family are equipped with an orthogonal model of persistence. The issues related to the model of persistence are discussed in section 3.

In section 4 we introduce a strictly more powerful model supporting parametric polymorphism. In this section we also analyze the implications of this feature for object-oriented database technology.

In section 5 we show that the ODMG Standard requires a model with reflective capabilities. In this section we introduce the reflective model of the family of the ODMG Object Models.

In section 6 we present the most sophisticated model in the family. The most sophisticated model integrates an advanced object-oriented type system with a logic-based constraint language. Declarative constraint capabilities are completely missing in the ODMG Standard.

In section 7 we provide a summary of the analysis presented in the paper.

2 The Java Model

The basic model in this family corresponds to Java^{TM} [12] and the Java binding of the ODMG Standard [10]. Full compatibility with Java is the major advantage of this model, with a number of obvious implications. However, this model also has non-trivial limitations in database systems. These limitations are a consequence of the limitations of the Java type system.

Consider an abbreviated Java interface which specifies a generic collection type:

```
public interface Collection
{ public boolean add(Object obj);
  public boolean contains(Object obj);
  public boolean isEmpty();
  public Iterator iterator();
  public boolean remove(Object obj);
  public int size();
  ...
}
```

The ODMG `DCollection` interface extends the Java `Collection` interface with a variety of self-explanatory query methods.

```
public interface DCollection  extends Collection
{ public boolean existsElement(String predicate);
  public DCollection query(String predicate);
  public Iterator select(String predicate);
  public Object selectElement(String predicate);
}
```

Specific ODMG collection types `DSet`, `DBag`, `DList`, and `DArray` extend the interface `Collection`. The interface `DBag` is given below.

```
public interface DBag  extends DCollection
{ public DBag union(DBag otherBag);
  public DBag intersection(DBag otherBag);
  public DBag difference(DBag otherBag);
  public int occurrences(Object obj);
}
```

The only way to specify a generic collection type in the Java type system is to specify that the elements are of type `Object`. This fact has far reaching implications. Static typing of collections and processing of collections with static type checking is impossible. Extensive dynamic type checking is unavoidable. Every time an object is retrieved from a collection, a type cast must be applied to it in order to perform specific actions on the object. Otherwise, only the methods from the class `Object` may be invoked on an object from a collection.

The generic specification of the interface `Iterator` given below in the Java model suffers from the same problem.

```
public interface Iterator
{ public boolean hasNext();
  public  Object next();
  public void remove();
}
```

Because of this, processing collections requires extensive and seemingly redundant type casts. This is illustrated by the following example:

```
Iterator it=employees.iterator();
while (it.hasNext()) {
 Employee e=(Employee)it.next();
...
}
```

These limitations of the Java model have non-trivial implications on type checking OQL queries. In order to illustrate the problem, consider a Java collection `employees`:

```
Collection employees;
```

Note that it does not help to use inheritance to derive `EmployeeCollection` from `Collection`. The signatures of the methods inherited from `Collection` would have to remain the same. This is yet another limitation of the Java type system. It is actually needed in order to guarantee type safety when dynamic binding of messages to methods is in place. Because of all this, the following simple OQL query fails a type check:

```
select x.name
from   x in employees
where  x.salary >= 50,000
```

The reason for the above query to fail type checking is that the range variable `x` in the above query stands for an element of the collection `employees`. But the type of elements of the collection `employees` is `Object`. `Object` is not equipped with attributes or methods `name` and `salary`.

Note that this OQL style of accessing features of objects requires that those features are declared public in the Java classes. From the object-oriented prospective, this would be justified if those features are public methods. The fact is that in object-oriented database systems (O_2 in particular), fields are often accessed directly. This violates the information hiding principles, but it makes object-oriented query languages closer to SQL. In addition, optimizing queries that invoke only public methods is a major issue.

In order to make type checking of OQL queries succeed, type casts must be used. In the example below, such a type cast is applied to the range variable, so that the query now type checks.

```
select x.name
from   (Employee)x in employees
where  x.salary >= 50,000
```

Note that full accordance with Java would in fact require the type cast to be applied to every occurrence of the range variable `x`.

So in the simplest model, OQL queries must have the above, rather annoying, form. From the user's viewpoint, the query is awkward because the user is forced to specify a seemingly redundant type information. From the system viewpoint, type casts necessarily generate dynamic type checks. Such checks reduce run-time efficiency, especially in database systems, where large scale collections are typically accessed. In addition, reliability is also reduced, because if a type check fails, appropriate run-time mechanisms must be invoked. These are always more problematic in database systems. Indeed, if a query is embedded in a transaction, even recovery procedures may have to be invoked. This problem is general and does not apply to queries only. If a transaction is processing a collection using the interface `Iterator`, the same problems with the same implications will occur.

3 Orthogonal Persistence

All the models in this family are equipped with transparent, orthogonal, and transitive persistence [4]. Transparency means that the model of persistence frees the users from all low-level details of managing persistent objects. Orthogonality means that objects of any type may be persistent or transient. In fact, at any given point in time, there may exist both transient and persistent objects of any class. Transitive persistence guarantees proper and transparent management of complex objects. When an object is promoted to persistence, all its components, direct or indirect, are also promoted to persistence. This property is called reachability [4].

The Java binding of the ODMG Standard actually supports reachability, but not orthogonality. The point is that both are needed in order to manage properly complex persistent objects [1].

Although the ODMG Standard does not promote orthogonal persistence [1], the fact is that its `Database` class suggests orthogonal persistence. The `Database` class of the Java binding of the ODMG Standard is given below.

```
public class Database
{ ...
 public static Database open(String name, int accessMode)
                        throws ODMGException;
 public void close() throws ODMGException;
 public void bind(Object obj, String name);
 public Object lookup(String name)
               throws ObjectNameNotFoundException;
 public void unbind(String name)
             throws ObjectNameNotFoundException;
}
```

The method `bind` makes the first argument object persistent and associates it with the name given as the second argument. The receiver of this method is

a database object. Executing the method `bind` introduces this binding in the name space of the receiver database. In order to specify the method `bind` in a generic manner, the first argument must necessarily be of type `Object`. A natural conclusion would be that an object of any type may be promoted to persistence (orthogonality). By reachability, all direct and indirect components of the promoted object are also made persistent, and all of this is done in a transparent fashion.

Note that when a persistent object is looked up by its name invoking the method `lookup`, the result is necessarily of type `Object`. A type cast is thus necessary in order to execute any specific methods on the retrieved object.

The above described model has actually been implemented in the PJama system [5]. One of the advantages of PJama is that it requires no changes of the Java programming language in order to support orthogonal persistence. The functionalities comparable to the the features the ODMG `Database` class are encapsulated in the `PJStore` interface of PJama. However, an extension of the Java Virtual Machine is required. Since system changes are required, we have suggested a different approach [2], which is in fact more natural for Java. However, this proposal also requires some system changes.

The fact that Java has the root object type (and so do Smalltalk and Eiffel), suggests the most natural approach to extending Java with orthogonal persistence. The persistence capabilities should be associated with the class `Object`. Since all other classes necessarily extend the class `Object`, orthogonality of persistence is immediate.

This approach requires an extension of the class `Object` with a method that makes the receiver object persistent. The name provided as the parameter of the method `persists` is bound to the receiver object in the name space of the underlying database. A possible extension may also include a method which tests whether the receiver object is persistent. This is illustrated in the following extension of the Java class `Object`.

```
public class Object
{ ...
   public boolean equals(Object obj);
   public int hashCode();
   public Object clone();
   public final Class getClass();
// public final boolean isPersistent();
// public final void  persists(String name);
}
```

The model of persistence that associates persistence capabilities with the root class has been implemented for a declarative, temporal constraint language *MyT* [2]. The implementation has been carried out on top of the Java Virtual Machine using PJama as the lower level support for persistence. Note that this model is truly object oriented, since it is based entirely on message passing and inheritance.

Orthogonal persistence has not been adopted by the ODMG because it does create implementation problems. However, at this point there are two industrial persistent Java systems that support orthogonal persistence, save for some difficult cases, such as threads. One of them is PJama [5], a research project developed at SUN in cooperation with the University of Glasgow. The other one is a commercial system Gemstone$_J$ [9]. The latter is a true database application server.

The ODMG Standard is based on the notion of persistence capable classes. Objects of a persistence capable class may be persistent or transient. Objects of a class that is not persistence capable are always transient. In this model, persistence is a property of classes rather than objects. This is why the model of persistence in the ODMG Standard is not orthogonal. In an orthogonal model of persistence, all classes are persistence capable.

In order to illustrate the implications, suppose that the class `Employee` has an attribute `picture` of type `Image`. In a database of employees, the class `Employee` will naturally be persistent capable. However, `Image` is presumably a class of general interest, and it does not necessarily belong to the employee database. In fact, it is likely that `Image` belongs to a library developed without any considerations related to persistence. Hence, `Image` may not be persistence capable. The implication of this likely situation is that when an employee object is promoted to persistence, its picture will not persist.

Orthogonality has major implications on software reuse, which is the cornerstone of the object-oriented paradigm. In an orthogonal model of persistence, all libraries developed without persistence considerations in mind become reusable for database applications. For example, the library of Java collection interfaces becomes readily available for database applications. Our experiments include graphic and other packages developed without any considerations of persistence in mind.

The interface `Collection` given in section 2 is just a Java library type. `DCollection` is, however, an ODMG type. The implementation class for `DCollection` is presumably persistent capable. However, as far as Java libraries are concerned, the class implementing `Collection` is not persistence capable. Furthermore, `Object` is not a persistent capable class either. So elements belonging to a collection object of type `DCollection` cannot persist! This analysis demonstrates the controversies of the model based on persistence capable classes. It also shows the advantages of our proposed model of orthogonal persistence for Java, which associates persistence capabilities with the root class `Object`.

4 Parametric Polymorphism

A strictly more sophisticated model is obtained from the basic Java model by introducing parametric polymorphism. This is illustrated by the parametric interface `Collection<T>` given below. The interface has one type parameter which ranges over the set of all types. This is why this basic form of parametric polymorphism is called universal type quantification.

```
public interface Collection<T>
{ public boolean add(T obj);
  public boolean contains(T obj);
  public boolean isEmpty();
  public Iterator<T> iterator();
  public boolean remove(T obj);
  public int size();
  ...
}
```

Note that the iterator interface is now also parametric. So is the interface `DCollection<T>` which extends the parametric interface `Collection<T>`.

```
public interface DCollection<T>
                 extends Collection<T>
{ public boolean existsElement(String predicate);
  public DCollection<T> query(String predicate);
  public Iterator<T> select(String predicate);
  public T selectElement(String predicate);
}
```

This model brings major advantages to object-oriented database technology, in comparison with the basic Java model [1]. It is now possible to statically type collections, so critical for any database technology. The same applies to programs that operate on collections, database transactions in particular. All collection types become parametric, as well as many other types relevant for an object-oriented database technology. A particularly important example of such a type is the iterator type in the `Collection<T>` interface.

The generic (parametric) interface `Iterator<T>` is given below:

```
public interface Iterator<T> {
 public boolean hasNext();
 public T next();
 public void remove();
}
```

Instantiating parametric types is carried out at compile time, such as in the example of the `employees` collection given below. Note that the specific result type of the query methods `query`, `select` and `selectElement` will now be determined at compile time.

```
Collection<Employee> employees;
```

It is now possible to process collections with no type casts. Static type checking is now possible, as illustrated by the following example:

```
Iterator<Employee> it=employees.iterator();
while (it.hasNext()) {
 Employee e=it.next();
...
}
```

Note that the code is now more natural for the users, as it does not contain type casts.

In the model with parametric polymorphism, static type checking of OQL queries also becomes possible. The query given below, which fails type checking in the Java type system, now type checks. The reason is that it is now possible to infer at compile time the specific type of the range variable `x`.

```
select x.name
from   x in employees
where  x.salary >= 50,000
```

This is now indeed the form of a query as it appears in OQL. It is also the form of a query that SQL users are accustomed to. The inferred type of the range variable `x` is now `Employee`, because the type of the collection `employees` is now `Collection<Employee>`.

Universal type quantification is available in C++ via templates. This form of parametric polymorphism is not sufficiently powerful for the object-oriented database technology. Typing ordered collections requires the form of parametric polymorphism called constrained genericity [17], or bounded type quantification. This is illustrated by the following example:

```
public interface Comparable {
 ...
 public boolean lessThan(Comparable obj);
}

public interface OrderedCollection<T extends Comparable>
                        extends Collection<T> {
...
}
```

The parametric interface `OrderedCollection<T>` extends the parametric interface `Collection<T>`. However, it also puts a constraint on the type parameter `T`. This is why this form of parametric polymorphism is called bounded type quantification. The corresponding actual type parameter to be substituted for `T` must extend the interface `Comparable`. This means that the actual parameter must have the ordering methods such as `lessThan`, etc.

This form of parametric polymorphism is required in typing keyed collections and indices. In fact, an even more sophisticated form of parametric polymorphism, called F-bounded polymorphism, is preferable [1], [18]. It is particularly important to note that most published proposals for extending Java with parametric polymorphism include both bounded type quantification (constrained genericity) and even F-bounded polymorphism. A detailed analysis is given in [18].

5 Reflection

In more traditional database technologies queries and transactions are programs. Queries are programs expressed in a declarative language. Transactions are typically programs in a programming language with possibly embedded queries. The advantage of this well-established and widely used approach is that it makes possible mostly static type checking and optimization of queries and transactions. This, of course, makes a big difference in efficiency and reliability.

However, a reflective model is required in order to capture the view of queries and transactions in the ODMG Standard. Both are viewed as objects [10]. As such, they are created dynamically, i.e., at run-time. Being objects, transaction and query objects communicate via message passing, and thus execute methods. Proper limitations must be introduced in order not to defeat the whole purpose of the type system. In other words, the desired reflective model must be type safe.

In the reflective model transactions are objects of the class `Transaction`. This class is given below. It follows closely the `Transaction` class of the Java binding of the ODMG Standard, but the transaction model is actually quite different.

```
public class Transaction {
  public Transaction(Runnable code);
  public void begin();
  public void commit();
  public void abort();
  public void checkpoint();
  ...
}
```

The reflective nature of this model is manifested in the above class. Its constructor takes a runnable object as its parameter. This object is an instance of a class that contains the actual transaction code. This seemingly small difference in comparison with the ODMG Standard is actually a major difference in the transaction model. This feature makes the model proposed in this paper truly reflective, and tied with reflective capabilities of Java.

An alternative to the above `Transaction` constructor would be a constructor which takes an object of the class `Thread` as its parameter. The implication is that a newly created transaction object is in fact associated with a new thread. This, of course, is a very natural model. However, the model of locking in Java [12] presents limitations with respect to the requirements customary in database technologies. These issues remain open for further investigations.

In the reflective model queries are objects of the class `OQLQuery` given below. This is a class of the Java binding of the ODMG Standard [10].

Note that query methods are also a distinctive feature of the interface `DCollection`. In both cases OQL expressions are provided as string parameters of the appropriate methods, because a Java compiler would not recognize those as valid expressions.

Unlike techniques commonly used for embedded SQL, this technique does not require a preprocessor. However, static parsing, type checking, and optimization are completely lost, and must be done at run-time.

```
class OQLQuery {
  public OQLQuery();
  public OQLQuery(String query)
         throws QueryInvalidException;
  public void create(String query)
              throws QueryInvalidException;
  public void bind(Object parameter)
              throws QueryParameterCountInvalidException,
                     QueryParameterTypeInvalidException;
  public Object execute()
                throws QueryException;
}
```

It is interesting to point out that more traditional techniques based on a precompiler would actually allow compiled queries. For example, this is the case with SQL$_J$. As it is, queries in the Java biding of O_2 and the ODMG Standard must be either interpreted at run-time or dynamically compiled. The latter technique is in fact possible in the current Java technology, but it is not standard across different platforms.

An example from O_2 [16] of constructing dynamically a query object is given below. Once a query object is constructed, the actual parameters are passed to it by invoking the method `bind`. Note that the formal parameters are denoted as `$i`, where `i` is an integer. When a query is completed at run-time, the method `execute` is invoked on the completed query object. The result of query execution is of the type `Object`, and thus type casting is typically required.

```
DBag persons;
DBag employees;
...
OQLQuery query = new OQLQuery(‘‘ select e
                                from   e in employees
                                where  e.salary > $1 and e in $2’’);
...
Double salary = new Double(50000);
query.bind(salary);
query.bind(persons);
employees = (DBag)query.execute();
```

Providing a string where a boolean expression (a predicate in general) is naturally expected is a form of ad-hoc polymorphism which a conventional object-oriented type system cannot handle. An implementation technique that we have developed for this reflective model is run-time linguistic reflection. Run-time linguistic reflection was initially developed for languages that are not object-oriented [21]. When applied to the object-oriented technology, this technique

requires run-time generation of source code for classes. These classes are subsequently compiled at run-time and linked to the running classes.

The technique of run-time linguistic reflection depends on the availability of type information at run-time. Java Core Reflection is exactly what is required in this technique. In addition to providing run-time type information, Java Core Reflection also allows invocation of methods discovered in the process of run-time type introspection. Dynamic compilation is an essential component of this technique. This feature is not standard in the current Java technology, but it is in fact available [14].

6 Constraints

In the most advanced model in this family, an advanced type system is integrated with a logic-based constraint (assertion) language. Such constraint facilities are completely lacking in the ODMG Standard. Constraints and triggers are important for most database technologies, particularly those for active database management [11].

Among object-oriented programming languages, Eiffel [17] is a notable example of a language that has integrated assertions. C++ also has assertions as an add-on feature. Regrettably, Java is completely lacking such declarative language capabilities.

Object-oriented assertions include method preconditions and postconditions, class invariants, and history properties [15]. All of them are logical expressions. The expressibility of these assertions is a major issue. The simplest solution is to allow only boolean expressions of the underlying programming language, as in Eiffel and C++. More powerful approaches would allow Horn-clause form of assertions. Even the full-fledged first order predicate calculus would be desirable. However, the assertions of this level of expressiveness are problematic, because of the lack of a suitable implementation technology.

Enforcing statically even a simple form of assertions in a full-fledged procedural object-oriented language is a major problem. The reason is the complexity of the task with procedurally decomposed methods. The problem is, of course, only much more difficult with a more expressive constraint language, which typically requires universal and existential quantification. In Eiffel and C++, constraints of such a form are encoded into methods, and hence run-time checking of the assertions is unavoidable. Run-time exception handling is thus required if a constraint fails.

Static verification of database transactions [19] would, of course, be preferable. Static verification is actually possible for a more limited, higher-level transaction language [19]. This is one of the reasons for developing a high-level, object-oriented constraint language, integrated with an advanced type system, and implemented on top of a persistent extension of the Java Virtual Machine [2].

Method preconditions are assertions that are required to be satisfied before method execution. This is the responsibility of the clients. Method postconditions are assertions that specify the properties guaranteed by the class designer to hold

upon completion of method execution. Of course, this guarantee holds as long as the method precondition was satisfied before method execution.

Method preconditions and postconditions are illustrated below for a parametric interface `Collection<T>`. The `requires` clause specifies method precondition, and the `ensures` clause specifies method postcondition, as in [17] and [15].

```
public interface Collection<T>
{ public boolean contains(T obj);
  public int size();
  public boolean add(T obj)
    ensures this.contains(obj),
            this.size()=(old)this.size()+1;
  public boolean remove(T obj)
    requires this.contains(obj)
    ensures  this.size()=(old)this.size()-1;
  ...
}
```

The above interface contains also an assertion that is in fact a history property. History properties are constraints that specify allowed sequences of object states [15]. For practical purposes, assertions of this form are usually enforced for pairs of successive object states. The assertion `this.size()=(old)this.size()+1` illustrates this point. The keyword `this` denotes the receiver object in its current state. In a postcondition assertion, the object state is the state after method execution. The expression `(old)this` denotes the receiver object in the state prior to method execution. This form of referring to the previous object state using the keyword `old` is used in Eiffel [17].

Class invariants are constraints that must be satisfied by all object states outside of method execution [15]. If the proper information hiding discipline is exercised (which typically is not the case in object-oriented databases), this means that class invariants must hold for all object states visible to the clients. Class invariants are illustrated below for a parametric interface `Set<T>`. The interface `Set<T>` extends the previously given interface `Collection<T>` by two additional methods `union` and `intersection` along with the associated constraints, that are in fact class invariants.

```
public interface Set<T>
                 extends Collection<T>
{public Set<T> union(Set<T> S);
 public Set<T> intersection(Set<T> S);
 ...
 invariant:
 forAll Set<T> this, S; T: obj
 (this.union(S).contains(obj) <- this.contains(obj);
  this.union(S).contains(obj) <- S.contains(obj);
  this.intersection(S).contains(obj) <- this.contains(obj),
                                        S.contains(obj));
  ...
}
```

Note that the methods and the constraints of the interface `Collection<T>` are inherited in `Set<T>`. The logic paradigm used in the above example is Horn clause logic, which is the basis for logic programming languages. The symbol `<-` denotes implication and comma denotes conjunction. This logic was chosen because of its well-known nice properties: well-defined formal semantics (in which the monotonicity plays a critical role), and the existence of an execution model tied to the formal semantic.

However, Horn clause logic has its limitations. It cannot express negative information. This is why it would not be possible to specify the precondition assertion in the interface `Set<T>` which requires that the element to be inserted does not already belong to the receiver set. Likewise, the postcondition assertion which guarantees that the removed element does not belong to the receiver set, cannot be expressed in Horn clause logic.

The availability of assertions allows specification of semantics (or at least a part of it) in a declarative and formal manner. This way the behavioral properties of objects of a class are communicated to the clients with no need to investigate procedurally decomposed methods. This also allows addressing the issue of behavioral compatibility in the inheritance hierarchies.

The rules of the type system guarantee that when an object of type `Set<T>` is substituted where an object of the type `Collection<T>` is expected, type safety will be guaranteed. This holds for any particular type `T`. This means that an object will never get a message for which it does not have a method with a correct signature. But an object-oriented type system does not address the issue of method semantics.

In fact, in a typical, full-fledged object-oriented language (such as Java), it is possible to redefine the semantics of an inherited method in any way, as long as its signature satisfies the rules of the type system. This leads to all kinds of semantic incompatibilities. For example, suppose that in the interface `Set<T>` we specify the precondition for the method `add` which requires that the element to be added does not already belong to the receiver set. This creates an immediate behavioral incompatibility with the super interface `Collection<T>`. Indeed, collections include bags, and there is no such requirement for adding an element to a bag.

The formal rules for behavioral compatibility are expressed by the notion of behavioral subtyping [15]. These rules have a particular implementation in the programming language Eiffel [17]. While these issues have attracted some amount of attention in object-oriented programming languages, they have not been addressed at all in database research [8], [20]. The earlier results on static verification of database transactions [19] did not have to deal with the issue of dynamic binding of messages to methods. The reason is that the verification techniques were applied to the relational paradigm, rather than to object-oriented languages. This is precisely where the problem lies in static verification of object-oriented transactions. The subtleties of the rules for behavioral subtyping are precisely caused by dynamic binding.

The operator `old` used in Eiffel to refer to the object state prior to method execution is in fact a temporal operator. There are strong indications that a suitable temporal paradigm is quite promising as a logic basis for an object-oriented assertion (constraint) language. Class invariants implicitly involve yet another temporal operator: `always`. Indeed, class invariants are assertions that hold in all user-visible object states. Results on the design and implementation of a typed, temporal object-oriented constraint language are given in [2]. The relevance of these results are in part due to the fact that the implementation has been carried out on top of a persistent extension of the Java Virtual Machine (PJama).

7 Conclusions

We showed that a single ODMG Object Model cannot be a basis for the variety of languages covered by the ODMG Standard. A sufficient reason is that these languages have different type systems. We have developed an alternative approach based on a family of the ODMG Object Models.

The most basic model in this family corresponds to the Java programming language. This fact is its main advantage. Disadvantages of this model are caused by the fact that Java at the moment does not support parametric polymorphism. Because of this, in the Java model, static typing of collections and queries is not possible. Frequent and seemingly redundant type casts are required. This implies frequent dynamic type checks, which reduces efficiency and reliability.

All the models in the proposed family of the ODMG Object Models are equipped with a transparent model of persistence that supports orthogonality and reachability. We showed how such a model of persistence fits into the basic Java model. The features of the proposed model have major advantages over the model of persistence of the ODMG Standard, particularly in managing complex persistent objects, and making Java libraries reusable in persistent object systems.

A strictly more sophisticated model is obtained by extending the basic model with parametric polymorphism. We showed that the form of parametric polymorphism required for object-oriented database technology is in accordance with the published proposals for extending Java with parametric polymorphism. A significant advantage of the model with parametric polymorphism is that it allows static type checking of collections and queries.

We showed that the technology proposed by the ODMG Standard requires a reflective model as well. The features of such a model are specified in the paper. It was shown that it is possible to accommodate the required reflective features in the Java technology. The model is, of course, type safe. Java Core Reflection and dynamic compilation capabilities are essential components of the implementation technique developed for this model.

The most sophisticated model in the family is equipped with a logic-based constraint language. Such constraint capabilities are completely missing from the ODMG Standard. In this proposal the constraint language is integrated with an

advanced type system. The constraint language extends the expressive power of the proposed family in specifying behavioral, semantic properties of objects. This has implications on active database management, and static verification of object-oriented transactions.

The analysis presented in the paper also reveals the limitations of Java as a database programming language. Some of the required extensions do not affect the language, but require an extension of the underlying technology. This is the case with orthogonal persistence, which requires an extension of the Java Virtual Machine. In addition, our proposal for persistence in the root class requires extending the class `Object` with additional methods. Linguistic reflective capabilities require that run-time compilation capabilities are fully supported, and standard across different platforms.

Extending Java with parametric polymorphism is absolutely essential for database technologies. It requires changes in the language, but also non-trivial modifications in the underlying technology [18]. Extending Java with constraints is the most serious change in the language. The implementation technology suitable for database systems is also a major extension of the existing Java technology.

There exists a substantial amount of formal development related to the material presented informally in this paper. The availability of the formal foundation for the family of the ODMG Object Models is an essential property of the overall approach. Such a formal definition, not to speak of a theory, is simply not possible for the ODMG Object Model, because of its inconsistencies. The formal theory underlying the family of the ODMG Object Models is necessarily the topic of a separate paper. The most distinctive feature of this formal development is that it integrates the relevant formal results on object-oriented type systems with a formally established object-oriented constraint language.

Acknowledgement

Svetlana Kouznetsova contributed useful comments on the final version of this paper.

References

1. S. Alagić, The ODMG Object Model: Does it Make Sense?, Proceedings of the OOPSLA '97 Conference, pp. 253-270, ACM, 1997. 15, 15, 18, 18, 21, 22
2. S. Alagić, J. Solorzano, and D. Gitchell, Orthogonal to the Java Imperative, Proceedings of ECOOP '98, *Lecture Notes in Computer Science*, *1445*, pp. 212-233, 1998. 19, 19, 25, 28
3. S. Alagić, O_2 and the ODMG Standard: Do They Match?, *Theory and Practice of Object Systems*, to appear, 1999.
4. M. Atkinson and R. Morrison, Orthogonally Persistent Object Systems, *VLDB Journal 4*, pp. 319-401, 1995. 18, 18

5. M. Atkinson, L. Daynes, M. J. Jordan, T. Printezis, and S. Spence, An Orthogonally Persistent Java^{TM}, ACM SIGMOD Record *25*, pp. 68-75, ACM, 1996. 19, 20
6. M. Atkinson, F. Bancilhon, D. DeWitt, K. Dittrich, and S. Zdonik, The Object-Oriented Database System Manifesto, Proceedings of the First Object-Oriented and Deductive Database Conference (DOOD), pp. 40-75, Kyoto, 1989.
7. F. Bancilhon, C. Delobel, and P. Kanelakis, *Building an Object-Oriented Database System: The Story of O_2*, Morgan Kaufmann Publishers, 1993.
8. V. Benzaken and D. Doucet, Themis: A Database Language Handling Integrity Constraints, *VLDB Journal 4*, pp. 493-517, 1994. 27
9. B. Bretl, A. Otis, M. San Soucie, B. Schuchardt, and R. Venkatesh, Persistent Java Objects in 3 Tier Architectures, in: R. Morrison, M. Jordan, and M. Atkinson: *Advances in Persistent Object Systems*, pp. 236-249, Morgan Kaufmann Publishers, 1999. 20
10. R. G. G. Cattell, D. Barry, D. Bartels, M. Berler, S. Gamerman, D. Jordan, A. Springer, H. Strickland, and D. Wade, *The Object-Oriented Database Standard: ODMG-2.0*, Morgan Kaufmann, 1997. 14, 16, 23, 23
11. N. Gehani and H. V. Jagadish, Ode as Active Database: Constraints and Triggers, Proceedings of the VLDB Conference, pp. 327-336, Morgan Kaufmann, 1991. 25
12. J. Gosling, B. Joy, and G. Steele, *The Java^{TM} Language Specification*, Addison-Wesley, 1996. 16, 23
13. M. Jordan and M. Atkinson, Orthogonal Persistence for Java - A Mid-term Report, in: R. Morrison, M. Jordan, and M. Atkinson: *Advances in Persistent Object Systems*, pp. 335 - 352, Morgan Kaufmann Publishers, 1999.
14. G. Kirby, R. Morrison, and D. Stemple, Linguistic Reflection in Java, *Software Practice and Experience 28*, *10*, 1998. 25
15. B. Liskov and J. M. Wing, A Behavioral Notion of Subtyping, *ACM Transactions on Programming Languages and Systems 16*, pp. 1811-1841, 1994. 25, 26, 26, 26, 27
16. O_2 Technology, ODMG Java Binding User Manual, Release 5.0, Ardent Software, 1998. 24
17. B. Meyer, Eiffel: *The Language*, Prentice-Hall, 1992. 22, 25, 26, 26, 27
18. J. Solorzano and S. Alagić, Parametric Polymorphism for Java^{TM}: A Reflective Solution, Proceedings of OOPSLA '98, pp. 216-225, ACM, 1998. 22, 22, 29
19. T. Sheard and D. Stemple, Automatic Verification of Database Transaction Safety, *ACM Transactions on Database Systems 14*, pp. 322-368, 1989. 25, 25, 27
20. D. Spelt and H. Balsters, Automatic Verification of Transactions on an Object-Oriented Database, in: S. Cluet and R. Hull (Eds.), Database Programming Languages, *Lecture Notes in Computer Science 1369*, pp. 396-412. 27
21. D. Stemple, R. B. Stanton, T. Sheard, P. Philbrow, R. Morrison, G. N. C. Kirby, L. Fegaras, R. L. Cooper, R. C. H. Connor, M. Atkinson, and S. Alagić, Type-Safe Linguistic Reflection: A Generator Technology, ESPRIT Research Report CS/92/6, Department of Mathematical and Computational Sciences, University of St. Andrews, 1992, in: M. P. Atkinson (ed.), *The FIDE Book*, Springer-Verlag, 1999, to appear.

Consistent Handling of Integrity Constraints and Extensional Assertions for Schema Integration

Can Türker and Gunter Saake

Otto-von-Guericke-Universität Magdeburg, ITI
Postfach 4120, D–39016 Magdeburg, Germany
{tuerker,saake}@iti.cs.uni-magdeburg.de

Abstract. Schema integration is a main step in the database integration process. Since the semantics of a schema is also determined by its integrity constraints, a correct and complete schema integration has to deal with integrity constraints existing in the different schemas to be integrated. In this paper, we present a framework for the correct handling of integrity constraints during schema integration. In particular, we work out the correspondence between local integrity constraints and global extensional assertions. The knowledge about the correspondences between the underlying integrity constraints and extensional assertions can then be utilized for an augmented schema integration process.

1 Introduction

Nowadays, large organizations are faced with the problem of integrating data stored in independently developed and possibly heterogeneous database systems [13]. Especially, for decision support correct and "up-to-date" data coming from different databases is essential. Schema integration is the main step in the database integration process (for a survey see [2]). Schema integration aims at deriving an integrated schema which is a virtual view on all database classes to be integrated [9,7]. An integrated schema provides global users a uniform and transparent interface to distributed and possibly heterogeneous databases.

A key point in schema integration is resolving existing semantic heterogeneity [5] between the schemas to be integrated. Semantic heterogeneity occurs when one real-world concept is modeled by database designers in different ways. One form of semantic heterogeneity are differences in the integrity constraints of the modeled object classes. The detection and handling of such constraint differences is fundamental for a correct schema integration. Another form of semantic heterogeneity occurs as conflicts in the extensional relationships among the classes of the schemas to be integrated. Extensional relationships are generally specified as assertions which are actually global integrity constraints.

The database integrator is supposed to be an expert of the application domain who knows about the semantics of the schemas to be integrated. Obviously, it is a hard task for the database integrator to get a grasp of all integrity constraints existing in the different local schemas. Nevertheless, the precise specification of local integrity constraints and global extensional relationships is a prerequisite for a semantically correct integration. Since integrity constraints are an integral part of a schema, they *must* be encompassed by a schema integration method.

Eder et al. (Eds.): ADBIS'99, LNCS 1691, pp. 31–45, 1999.
© Springer-Verlag Berlin Heidelberg 1999

Most existing integration methods, e.g. [10,8,14,4], do not consider integrity constraints at all. There are a few approaches, e.g. [11,1,16], which provide means for formulating global integrity constraints during the derivation of a global schema. However, these approaches consider integrity constraints independently of extensional assertions. In consequence, contradictions between the local integrity constraints and global extensional assertions are not excluded.

In this paper, we work out the correspondence between integrity constraints and extensional assertions and present an enhanced approach to schema integration which considers local integrity constraints as well as their relationship to global extensional assertions. We introduce the notion of base integrity constraints as a basic principle for a correct and complete schema integration.

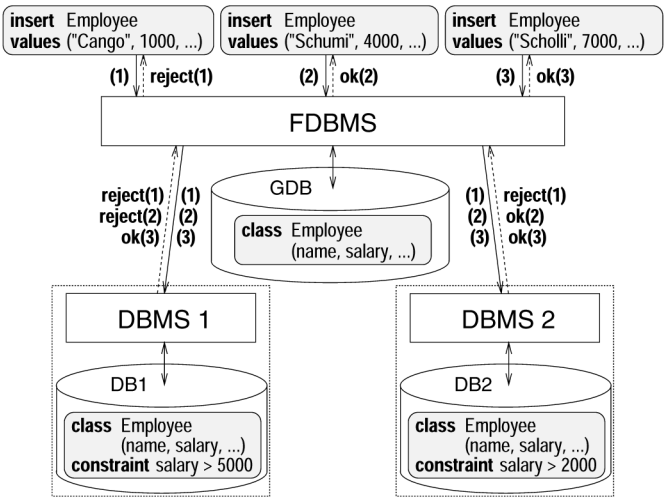
The paper is organized as follows: Sect. 2 provides the motivation for considering integrity constraints during schema integration. After presenting some preliminaries in Sect. 3, we work out the correspondence between integrity constraints and extensional assertions in Sect. 4. Sect. 5 introduces the notion of base integrity constraints and Sect. 6 sketches our extended integration approach.

2 Global and Local Understandability

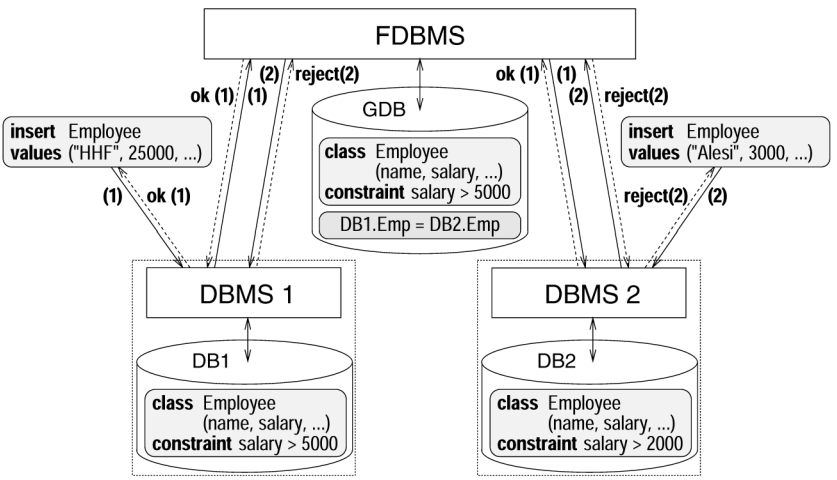
As pointed out in [2], schema integration has to fulfill the *completeness* and *correctness* requirements, i.e., an integrated schema should reflect the correct real-world semantics of the different local schemas involved. ¿From the correctness and completeness requirements immediately follows that the *information capacity* [6] of the global schema has to be equal to the union of the information capacities of the local schemas involved. Each local object (corresponding to a local schema which is part of the integrated schema) has to be representable in the integrated schema. On the other hand, a globally created object has to be representable by at least one local schema, i.e., a global object has to be mapped onto one or more objects of the corresponding local schemas.

We introduce the notion of *understandability* which means that the reason for the rejection of a database modification must be derivable from the integrity constraints of the schema. *Global understandability* demands that a global transaction should not be rejected if it fulfills all global integrity constraints. Analogously, *local understandability* requires that a local transaction should not be rejected if it satisfies all integrity constraints of the corresponding local schema.

The problem of global understandability is illustrated in the next figure. There are two related local classes Employee which are integrated into one global class Employee. The local class Employee managed by DBMS1 (DBMS2) contains an integrity constraint which defines that the attribute salary must have a value greater than 5000 (2000). However, the global schema does not reflect the local integrity constraints. Thus, the second and third global insertion requests can be successfully performed by the federated database management system, whereas the first global insertion request is rejected without any reasonable hint for the global user. From the global user's perspective there is no obvious explanation for the rejection of his request (because there are no integrity constraints on the global schema). The consequence is that the global user will send the same request several times until he gives up without any success.



The problem of local understandability is illustrated in the figure below. In traditional integration approaches, the database integrator specifies how the classes of the different schemas are extensionally related to each other. In our example, there is a global constraint specifying that the extension of the class DB1.Employee must always be equivalent to the extension of the class DB2.Employee. If this global constraint is violated by a local operation, this operation has to be propagated to the second local database system. Due to the *transparency* requirement, the propagation has to be *non-visible* for the local user, i.e., the local user should not aware that "his" local database system is involved in a federated database system. However, in our example the local insertion into DB2.Employee is rejected in order to maintain the global integrity constraint that both classes must be extensionally equivalent. In this case, a "non-visible" propagation of the insertion is not possible because the object to be inserted does not fulfill the constraints of the class DB1.Employee. Hence, the local insertion is rejected without any obvious reason for the local user.



From the motivating examples above we can state the following theses:

1. Integrity constraints of the local schemas must be reflected in the global schema (in order to avoid the problem of global understandability).
2. Global integrity constraints such as extensional assertions which are defined by the database integrator must be reflected in the local schemas (in order to avoid the problem of local understandability). That is, extensional assertions must correspond to local integrity constraints.

However, most existing approaches to schema integration [10,8,14,4] neither consider integrity constraints nor handle the connection between local integrity constraints and global extensional assertions.

3 Extensional Assertions and Constraint Relationships

A database object schema consists of a set of attributes of a given domain. The state of an object is determined by the values of its attributes. (Here, the terms object/tuple and class/relation are used interchangeable.) The object state space is defined by the Cartesian product of domains of the attributes. Objects with the same attribute sets are grouped into classes. The state or extension of a class is thus determined by its currently existing objects and their states.

Definition 1. For two classes C_1 and C_2 we define the following extensional assertions (Ext^t_C denotes the extension of class C at time t):

$$\begin{aligned}
\textit{Disjointness}: &\quad C_1 \varnothing C_2 :\Leftrightarrow \forall t\colon Ext^t_{C_1} \cap Ext^t_{C_2} = \emptyset \\
\textit{Equivalence}: &\quad C_1 \equiv C_2 :\Leftrightarrow \forall t\colon Ext^t_{C_1} = Ext^t_{C_2} \\
\textit{Containment}: &\quad C_1 \supset C_2 :\Leftrightarrow \forall t\colon Ext^t_{C_1} \supseteq Ext^t_{C_2} \\
\textit{Overlap}: &\quad C_1 \Cap C_2 :\Leftrightarrow \mathsf{true}
\end{aligned}$$

These assertions base on the notion of *semantic equivalence*, i.e., two objects refer to the same real-world object if their real world states are equal [10]. Thus, two classes are *extensionally equivalent* if and only if their extensions always correspond to the same set of real world objects. The other assertions have to be read analogously. □

A state space can be restricted by constraints. For example, the state space of an integer attribute salary can be restricted by a constraint (salary $>$ 2000). Constraints that can be checked on a single object are called *intra-object constraints*. *Inter-object constraints* can only be checked by considering at least two objects.

Definition 2. A *set of integrity constraints* Φ defines the admissible states of a database. We use the term *integrity constraint set* to refer to such a set. From a logical point of view, an integrity constraint set is a conjunction of integrity constraints. An integrity constraint itself is a formula in predicate logic. □

Definition 3. An integrity constraint set Φ_1 is *satisfiable* if and only if there exists a database state in which Φ_1 evaluates to true. An integrity constraint Φ_1 *implies* another integrity constraint Φ_2, written $(\Phi_1 \Rightarrow \Phi_2)$, if and only if all database states that satisfy Φ_1 also satisfy Φ_2. The negation $\neg(\Phi_1 \Rightarrow \Phi_2)$ states that there exists a database state that satisfies Φ_1 but not Φ_2. □

Definition 4. Let Φ_1 and Φ_2 be two *satisfiable* sets of integrity constraints. Then, following relationships between Φ_1 and Φ_2 are possible:

$$\begin{aligned}
\textit{Disjointness}\colon\ & \Phi_1 \varnothing \Phi_2 :\Leftrightarrow (\Phi_1 \Rightarrow \neg\Phi_2)\\
\textit{Equivalence}\colon\ & \Phi_1 \equiv \Phi_2 :\Leftrightarrow (\Phi_1 \Rightarrow \Phi_2) \wedge (\Phi_2 \Rightarrow \Phi_1)\\
\textit{Containment}\colon\ & \Phi_1 \supset \Phi_2 :\Leftrightarrow (\Phi_1 \Rightarrow \Phi_2) \wedge \neg(\Phi_2 \Rightarrow \Phi_1)\\
\textit{Overlap}\colon\ & \Phi_1 \Cap \Phi_2 :\Leftrightarrow \neg((\Phi_1 \varnothing \Phi_2) \vee (\Phi_1 \equiv \Phi_2) \vee (\Phi_1 \subset \Phi_2) \vee (\Phi_1 \supset \Phi_2))
\end{aligned}$$

Since disjointness is symmetric, we can also express it by $(\Phi_2 \Rightarrow \neg\Phi_1)$. □

The *constraint relationship problem* is to compute the relationship between sets of integrity constraints. Since the implication problem is undecidable for general constraints, we can state that the constraint relationship problem is also undecidable in the general case. Therefore, we have to restrict ourselves to constraint classes for which we can guarantee that this problem is solvable. In this paper, we concentrate on dense-order constraints (as example for intra-object constraints) and uniqueness constraints (as example for inter-object constraints):

1. Dense-order constraints are quantifier-free formulas of the form $(\mathsf{x}_1\,\theta\,\mathsf{c})$ and $(\mathsf{x}_1\,\theta\,\mathsf{x}_2)$, where x_1, x_2 are variables (attributes), c is a constant, and θ is one of $\{<, \leq, =, \neq, \geq, >\}$. The domain of the variables is assumed to be a countable infinite set (e.g., the real numbers) with a natural total order.
2. Uniqueness constraints $\textbf{unique}(\mathsf{x}_1, ..., \mathsf{x}_n)$ are formulas of the form

$$(\forall \mathsf{o}_1)(\forall \mathsf{o}_2)(\mathsf{o}_1.\mathsf{x}_1 = \mathsf{o}_2.\mathsf{x}_1 \wedge ... \wedge \mathsf{o}_1.\mathsf{x}_n = \mathsf{o}_2.\mathsf{x}_n \Rightarrow \mathsf{o}_1 = \mathsf{o}_2)$$

   where o_1 and o_2 are objects of a universe of discourse and $\mathsf{x}_1, ..., \mathsf{x}_n$ are attributes of these objects.

These classes are chosen only for illustration purposes. Please note that constraint relationship problem is (efficiently) decidable for a much larger class of constraints. An analysis of these constraints is not the subject of this paper.

In [15] we have presented a set of rules to determine the relationship between two dense-order constraints as well as between two uniqueness constraints. Using these rules, the relationship between integrity constraint sets consisting of dense-order and/or uniqueness integrity constraints can be computed in polynomial time. Here, we do not want to recall these rules; we just present some examples to illustrate these rules. Suppose the following integrity constraints are given:

$$\begin{aligned}
\Phi_1 &= \{(\mathsf{salary} > 2000)\}\\
\Phi_2 &= \{(\mathsf{salary} < 4000)\}\\
\Phi_3 &= \{(\mathsf{salary} > 5000)\}
\end{aligned}$$

The relationship between Φ_1 and Φ_2 is overlap. The constraint Φ_3 contains Φ_1 since all database states satisfying Φ_3 also satisfy Φ_1. Φ_2 and Φ_3 are disjoint since there exists no database state that satisfies both constraints.

If **unique**(ssn) holds on a class Employee, then **unique**(ssn, name) holds, too. The constraints **unique**(ssn) and **unique**(name, salary) overlap because the sets of employee objects restricted by these constraints overlap. The same argument holds for **unique**(ssn, name) and **unique**(name, salary).

4 Deriving Global Extensional Assertions

Traditionally, extensional assertions are fixed by the database integrator. Our goal is now to find out the precise correspondence between integrity constraints and extensional assertions. Given a set of local integrity constraints we want to conclude whether a certain extensional assertion is reasonable or not.

Since local schemas are modeled by different designers, the representation of the modeled real world may differ in the local databases. Therefore, it is important for the designer of a federated database to exactly know about the semantics of the local schemas. Here, several approaches are possible:

1. Assume that each local schema is correct and complete. That is, the integrity constraints of the local schemas are formulated as precise as possible. In this case, the integrity constraint sets allow only database states which correspond to possible real world states.
2. Assume that each local schema is correct but some of them may be incomplete with respect to the integrity constraint sets. This may occur if some integrity constraints are either missing or formulated too weak. In this case, database states are allowed for which there exists no correspondence in the real world.
3. Assume that a local schema may be incorrect in the sense that it is too restrictive. It does not allow database states which, however, may correspond to possible real world states. That is, there may occur states in the real world which cannot be represented by the local schema.

Due to space restrictions, we will concentrate on the first two approaches.

Correct and Complete Integrity Constraint Sets. Following the first approach, we exploit local integrity constraints to decide how two classes are extensionally related. The extensional relationship is determined by the local constraints.

Theorem 1. For two classes C_1 and C_2 with the *complete* integrity constraint sets Φ_{C_1} and Φ_{C_2} the following correspondences hold:

$$
\begin{aligned}
&(\Phi_{C_1} \varnothing \Phi_{C_2}) \Rightarrow (C_1 \varnothing C_2)\\
&(\Phi_{C_1} \equiv \Phi_{C_2}) \Rightarrow (C_1 \equiv C_2) \vee (C_1 \Cap C_2)\\
&(\Phi_{C_1} \supset \Phi_{C_2}) \Rightarrow (C_1 \subset C_2) \vee (C_1 \Cap C_2)\\
&(\Phi_{C_1} \Cap \Phi_{C_2}) \Rightarrow (C_1 \Cap C_2)
\end{aligned}
$$

Thus, we can imply that the extensions of the two classes are disjoint if their integrity constraint sets are disjoint. Analogously, from two overlapping integrity

constraint sets we can infer that the corresponding class extensions can only be specified as overlapping. In the other three cases, there are two possible derivations. □

From Theorem 1 follows that the database integrator must only intervene in case the integrity constraint sets are equivalent or in a containment relationship. In these cases, the designer has to choose between only two possibilities.

Example 1. Suppose, in a class C1 there is a constraint restricting the attribute salary to a value less than 5000 whereas in the another class C2 the "semantically corresponding" attribute salary must be greater than 5000. Then, there cannot exist a real world object which is member of both classes. On the other hand, if the integrity constraint sets of both classes C1 and C2 are equivalent, e.g. both consist of the constraint (salary $>$ 5000), then we can infer that the classes must be either extensionally equivalent or overlapping. In this case, the designer has to make a choice between the two possibilities. The other assertions are excluded by the assumption that the present integrity constraint sets are complete. ◘

Correct but Incomplete Integrity Constraint Sets. In general, the requirement that local integrity constraints have to be correctly and in particular completely formulated is very demanding. Usually, the local database designers do not capture all relevant integrity constraints for different reasons. Therefore, we have to deal with incomplete integrity constraint sets.

Theorem 2. For two classes C_1 and C_2 with the *incomplete* integrity constraint sets Φ_{C_1} and Φ_{C_2} the following correspondences hold:

$$(\Phi_{C_1} \varnothing \Phi_{C_2}) \Rightarrow (C_1 \varnothing C_2)$$
$$(\Phi_{C_1} \Theta \Phi_{C_2}), \Theta \in \{\equiv, \subset, \supset, \Cap\} \Rightarrow \bigvee_{\Omega \in \{\varnothing, \equiv, \subset, \supset, \Cap\}} (C_1 \, \Omega \, C_2)$$

In summary, from non-disjoint, incomplete integrity constraint sets we cannot derive any hint about the extensional assertions between the corresponding classes. In other words, the "disjoint" correspondence of Theorem 1 holds also for the second approach, whereas the others do not. □

The following example shall clarify the difference between the two approaches.

Example 2. Let there be two classes C_1 and C_2 with the following constraints:

$$\Phi_{C_1} = \{\textbf{unique}(\mathsf{salary}), (\mathsf{age} > 18)\}$$
$$\Phi_{C_2} = \{(\mathsf{age} > 18)\}$$

Obviously, Φ_{C_1} contains Φ_{C_2}. If we use the first approach to determine the extensional relationship between the classes C_1 and C_2, we obtain $C_1 \subset C_2$ and $C_1 \Cap C_2$ as correct extensional assertions. In contrast, in the second approach all possible extensional assertions are deducible. Therefore, a *semantic enrichment* step is required in order to complete the local integrity constraint sets:

- If the classes C_1 and C_2 describe disjoint sets of real world objects, for instance employees of different organizations, then this fact must be represented by a property (integrity constraint) which distinguishes the objects of the corresponding classes. In this case, add Φ_{1x} to Φ_{C_1} and/or Φ_{2y} to Φ_{C_2} such that $\Phi_{C_1} \varnothing \Phi_{C_2}$ holds. For example, Φ_{1x} may constrain the nationality of the employees of class C_1 to 'Turkish' whereas Φ_{2y} requires that the nationality of the employees of class C_2 is 'German'.
- If the classes C_1 and C_2 refer to the same set of real world objects, the Φ_{C_1} and Φ_{C_2} must be equivalent. If not, then this is an indication that at least one integrity constraint sets is too weak. In this case, add $\Phi_{C_1} \setminus \Phi_{C_2}$ to Φ_{C_2}.[1] In our example we have to add **unique**(salary) to Φ_{C_2}.
- If the class C_2 is a subclass of C_1 in the real world, then this fact must also be expressed by the relationship between Φ_{C_2} and Φ_{C_1}, i.e., there must exist at least one constraint in class Φ_{C_2} which is not in Φ_{C_1}. Here, we have to add $\Phi_{C_1} \setminus \Phi_{C_2}$ and another integrity constraint Φ_{2y} to Φ_{C_2}. In our example we add **unique**(salary) and e.g. (age $<$ 50) to Φ_{C_2}.

In conclusion, a semantic schema integration which intends to derive a correct and complete integrated schema requires correct and complete input informations. However, due to the inherent complexity of the problem this is not possible in each case. Therefore, sometimes we have to deal with incomplete informations and integrated schemas which do not exactly correspond to the modeled real world. In this case a semantic enrichment step is introduced to refine the corresponding local integrity constraint sets as far as needed.

5 The Concept of Base Integrity Constraints

When we consider several integrity constraints together we can divide these sets into disjoint partitions, called *base integrity constraints*. Base integrity constraints play a central role in deriving correct integrated schemas.

Definition 5. The *base intra-object constraint set* of an integrity constraint sets $\Phi_1, ..., \Phi_m$ is defined by a boolean formula in disjunctive canonical form basing on the intra-object constraints $\Psi_1, ..., \Psi_n$ of Φ_1, ..., Φ_m, where the term $\overline{\Psi_1}\,\overline{\Psi_2} \ldots \overline{\Psi_n}$ as well as all contradictory and redundant terms are excluded. A minterm in such a formula denotes a *base intra-object constraint*. □

The notion of base intra-object constraint set shall be illustrated by the following example. Suppose, there are the following three integrity constraints sets:

$$\Phi_1 = \{(\text{age} < 25), (\text{salary} > 2000)\}$$
$$\Phi_2 = \{(\text{age} > 18), (\text{salary} < 4000)\}$$
$$\Phi_3 = \{(\text{age} > 21), (\text{age} < 65), (\text{salary} > 2000)\}$$

Since the three integrity constraint sets are maximal overlapping, i.e., each possible intersection of these constraint sets is non-empty, we obtain the maximal number $(2^3 - 1)$ of possible base intra-object constraints (see next table).

[1] $\Phi_1 \setminus \Phi_2 :\Leftrightarrow \{\Phi_i \,|\, \Phi_i \in \Phi_1 \wedge (\nexists \Phi_j \in \Phi_2((\Phi_i \equiv \Phi_j) \vee (\Phi_i \subset \Phi_j))\}$.

In order to capture inter-object constraints like uniqueness constraints, we extend base intra-object constraint sets to *base integrity constraint sets*. The following example illustrates the steps to obtain a base integrity constraint set. Suppose, the following integrity constraint sets are given:

$$\Phi_1 = \{(\mathsf{age} < 25), \textbf{unique}(\mathsf{salary})\}$$
$$\Phi_2 = \{(\mathsf{age} > 18), \textbf{unique}(\mathsf{name})\}$$

Base Intra-Object Constraints		
1	$\Psi_1 \overline{\Psi_2}\,\overline{\Psi_3}$	$(\mathsf{age} \leq 18) \wedge (\mathsf{salary} \geq 4000) \vee (\mathsf{age} \leq 21) \wedge (\mathsf{salary} \geq 4000)$
2	$\Psi_1 \Psi_2 \overline{\Psi_3}$	$(18 < \mathsf{age} \leq 21) \wedge (2000 < \mathsf{salary} < 4000)$
3	$\overline{\Psi_1} \Psi_2 \overline{\Psi_3}$	$(18 < \mathsf{age}) \wedge (\mathsf{salary} \leq 2000) \vee (\mathsf{age} \geq 65) \wedge (\mathsf{salary} < 4000)$
4	$\overline{\Psi_1} \Psi_2 \Psi_3$	$(25 \leq \mathsf{age} < 65) \wedge (2000 < \mathsf{salary} < 4000)$
5	$\Psi_1 \Psi_2 \Psi_3$	$(21 < \mathsf{age} < 25) \wedge (2000 < \mathsf{salary} < 4000)$
6	$\Psi_1 \overline{\Psi_2} \Psi_3$	$(21 < \mathsf{age} < 25) \wedge (\mathsf{salary} \geq 4000)$
7	$\overline{\Psi_1}\,\overline{\Psi_2} \Psi_3$	$(25 \leq \mathsf{age} < 65) \wedge (\mathsf{salary} \geq 4000)$

Since the intra-object constraints of these sets are overlapping, we obtain three base intra-object constraints:

$$\Psi_1 \overline{\Psi_2} = \{(\mathsf{age} \leq 18)\}$$
$$\Psi_1 \Psi_2 = \{(18 < \mathsf{age} < 25)\}$$
$$\overline{\Psi_1} \Psi_2 = \{(\mathsf{age} \geq 25)\}$$

The base intra-object constraint set $\Psi_1 \overline{\Psi_2}$ is extended to a base integrity constraint by regarding the uniqueness constraint **unique**(salary). Analogously, $\overline{\Psi_1} \Psi_2$ is extended by adding the uniqueness constraint **unique**(name). The base intra-object constraint $\Psi_1 \Psi_2$ has to be divided into three base integrity constraints since the corresponding uniqueness constraints overlap:

$$\Psi_1 \Psi_2 \Upsilon_1 \overline{\Upsilon_2} = \{(18 < \mathsf{age} < 25), \textbf{unique}(\mathsf{salary}), \neg(\textbf{unique}(\mathsf{name}))\}$$
$$\Psi_1 \Psi_2 \Upsilon_1 \Upsilon_2 = \{(18 < \mathsf{age} < 25), \textbf{unique}(\mathsf{salary}), \textbf{unique}(\mathsf{name})\}$$
$$\Psi_1 \Psi_2 \overline{\Upsilon_1} \Upsilon_2 = \{(18 < \mathsf{age} < 25), \neg(\textbf{unique}(\mathsf{salary})), \textbf{unique}(\mathsf{name})\}$$

Please note that in general the computation of a base integrity constraint set for a given set of integrity constraint sets is not feasible in polynomial time. For computing the complete base integrity constraint set for n integrity constraint sets we have to consider all combinations of the sets involved (except the case where all constraint sets are negated). In the worst case, there are $2^n - 1$ base integrity constraints. Hence, a correct and complete schema integration — which base on a correct and complete set of base integrity constraints — cannot be performed in polynomial time. The worst-case occurs when all input classes are maximally overlapping. However, usually most of the input classes are extensionally disjoint, i.e., contain disjoint integrity constraint set. Therefore, the resulting number of base integrity constraints can be reduced drastically.

6 Integrating Schemata with Integrity Constraints

In this paper, we follow the schema integration method proposed in [12]. In this approach, the schema integration process is divided into two steps. In the first step an extensional decomposition of the input classes into the corresponding base extensions is performed. The decomposition bases on the information about the extensional assertions which are given by the database integrator. Based on these base extensions an integrated schema is then derived in the second step.

We extend this approach by exploiting the connection between extensional assertions and integrity constraints. Our idea is to derive correct extensional assertions among classes from different schemas by considering the corresponding sets of integrity constraints. As we have seen in Section 4, in some cases it is possible to precisely fix an extensional assertion, e.g. in case of disjoint integrity constraint set. In other cases, the relationship between the integrity constraint sets give hints about the possible, correct extensional assertions.

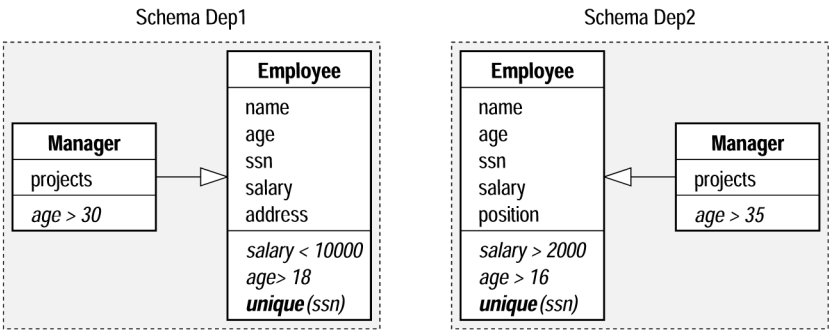
We propose the following steps to achieve a semantically correct and complete integrated schema:

1. As usual, in the first step the local schemas are transformed into a common data model and structural and semantic conflicts are resolved [2]. In particular, naming, type, and scaling conflicts are resolved.
2. Local integrity constraint sets are extracted from the local schemas. If possible, inter-object constraints are transformed into intra-object constraints. For instance, an inter-object constraint like $\mathbf{max}(x) < 20$ can be transformed into an intra-object constraint $(x < 20)$ (cf. Section 4.3 in [3]).
3. In the semantic enrichment step, any information implicit in the database schemas which was not explicitly expressed in the corresponding schemas are made explicit. In our context this means that local integrity constraint sets are extended by implicit constraints (see discussion in Section 3). In consequence, integrity constraints inherent to the schema such as specialization relationships are made explicit, e.g. by defining a "meta"-attribute class which constrains the names of the classes to which an objects belongs. In case of incomplete integrity constraint sets, the designer refines them until they correspond to the intended extensional relationships (see Section 4).
4. In this step the base integrity constraint set is derived from the local integrity constraint sets (see discussion in Section 5).
5. Each base integrity constraint is now assigned to a base extension.
6. Then, we derive an integrated schema following the algorithm in [12] on the basis of the base extensions derived in the previous step. Finally, as extension — in order to represent integrity constraint correctly — the integrity constraints of each class are computed from the base integrity constraints corresponding to the base extensions which are part of that class.

This approach is now illustrated by a small example. Suppose, there are two schemas as depicted in the figure below. Both schemas models employees and special employees, called managers. However, there are different constraints for employees and managers in the two schemas. In schema Dep1 employees must be older than 18 and managers older than 30. The salary of each employee must

be less than 10000. In schema Dep2 the employees must be older than 16 but the managers have to be older than 35. The salary of the employees must be higher than 2000. We assume that these integrity constraints are complete and correct.

In the following, we assume that the first three steps have been performed successfully. Structural and semantical conflicts are solved such that attributes with the same names correspond to the same real world concepts.



In step four the base integrity constraint set is derived. In our example scenario, there are the following four integrity constraint sets:

$$\Phi_{\mathsf{Dep1.Employee}} = \{(\mathsf{age} > 18), (\mathsf{salary} < 10000), \mathbf{unique}(\mathsf{ssn})\}$$
$$\Phi_{\mathsf{Dep1.Manager}} = \{(\mathsf{age} > 30), (\mathsf{salary} < 10000), \mathbf{unique}(\mathsf{ssn})\}$$
$$\Phi_{\mathsf{Dep2.Employee}} = \{(\mathsf{age} > 16), (\mathsf{salary} > 2000), \mathbf{unique}(\mathsf{ssn})\}$$
$$\Phi_{\mathsf{Dep2.Manager}} = \{(\mathsf{age} > 35), (\mathsf{salary} > 2000), \mathbf{unique}(\mathsf{ssn})\}$$

These integrity constraint sets result in seven base intra-object constraints (see following table).[2] Since the uniqueness constraint **unique**(ssn) is defined in all integrity constraint sets, the seven base intra-object constraints are extended to base integrity constraints by including this uniqueness constraint.

Base Intra-Object Constraints		
1	$\Psi_1\,\overline{\Psi_2}\,\Psi_3\,\overline{\Psi_4}$	$(18 < \mathsf{age} \leq 30) \wedge (2000 < \mathsf{salary} < 10000)$
2	$\Psi_1\,\overline{\Psi_2}\,\overline{\Psi_3}\,\overline{\Psi_4}$	$(18 < \mathsf{age} \leq 30) \wedge (\mathsf{salary} \leq 2000)$
3	$\Psi_1\,\Psi_2\,\overline{\Psi_3}\,\overline{\Psi_4}$	$(\mathsf{age} > 30) \wedge (\mathsf{salary} \leq 2000)$
4	$\Psi_1\,\Psi_2\,\Psi_3\,\overline{\Psi_4}$	$(30 < \mathsf{age} < 35) \wedge (2000 < \mathsf{salary} < 10000)$
5	$\Psi_1\,\Psi_2\,\Psi_3\,\Psi_4$	$(\mathsf{age} > 35) \wedge (2000 < \mathsf{salary} < 10000)$
6	$\overline{\Psi_1}\,\overline{\Psi_2}\,\Psi_3\,\Psi_4$	$(\mathsf{age} > 35) \wedge (\mathsf{salary} \geq 10000)$
7	$\overline{\Psi_1}\,\overline{\Psi_2}\,\Psi_3\,\overline{\Psi_4}$	$(16 < \mathsf{age} \leq 18) \wedge (\mathsf{salary} > 2000) \vee$ $(16 < \mathsf{age} \leq 35) \wedge (\mathsf{salary} \geq 10000)$

Since the original integrity constraints sets are divided into the base integrity constraints, they can be represented also as a union of the corresponding base

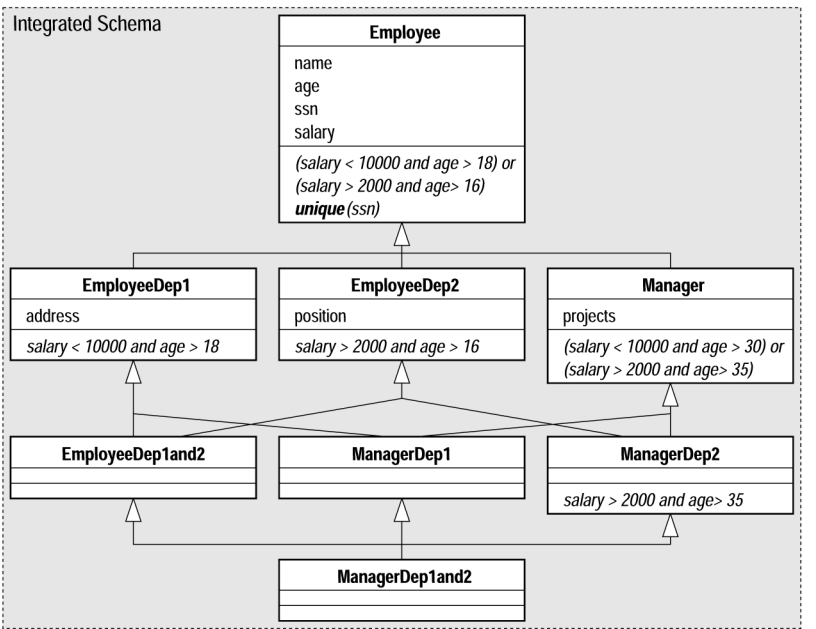
[2] $\Psi_1 = \Psi_{\mathsf{Dep1.Employee}}$, $\Psi_2 = \Psi_{\mathsf{Dep1.Manager}}$, $\Psi_3 = \Psi_{\mathsf{Dep2.Employee}}$, and $\Psi_4 = \Psi_{\mathsf{Dep2.Manager}}$.

integrity constraints. Thus, our example integrity constraints sets consist of the following base integrity constraints which are indicated by their numbers:

$$\Phi_{\mathsf{Dep1.Employee}} = 1 + 2 + 3 + 4 + 5$$
$$\Phi_{\mathsf{Dep1.Manager}} = 3 + 4 + 5$$
$$\Phi_{\mathsf{Dep2.Employee}} = 1 + 4 + 5 + 6 + 7$$
$$\Phi_{\mathsf{Dep2.Manager}} = 5 + 6$$

We now assign the base integrity constraints to the corresponding base extensions. Thus, there are seven base extensions denoted by the numbers 1–7.

These base extensions are now used as input for the schema integration algorithm described in [12]. The result derived by this algorithm is then extended by assigning the integrity constraints to each produced class. The final schema with the corresponding integrity constraints is depicted below.



In the following, we briefly sketch the way how the integrity constraints are attached to the classes of the integrated schema:

– Since the root class Employee consists of all seven base extensions, the integrity constraint set of this class consists of the logical disjunction of all seven base integrity constraints.

$$\Phi_{\mathsf{Employee}} = 1 \vee 2 \vee 3 \vee 4 \vee 5 \vee 6 \vee 7$$

– The class EmployeeDep1 consists of the base extensions 1–5:

$$\Phi_{\mathsf{EmployeeDep1}} = 1 \vee 2 \vee 3 \vee 4 \vee 5$$

– The class EmployeeDep2 is built from the base extensions 1, 4, 5, 6, and 7:

$$\Phi_{\mathsf{EmployeeDep2}} = 1 \vee 4 \vee 5 \vee 6 \vee 7$$

– The integrity constraint set of the class Manager yields from the base integrity constraints 3–6:

$$\Phi_{\mathsf{Manager}} = 3 \vee 4 \vee 5 \vee 6$$

– The class EmployeeDep1and2 consists of the base extensions 1, 4, and 5:

$$\Phi_{\mathsf{EmployeeDep1and2}} = 1 \vee 4 \vee 5$$

$\Phi_{\mathsf{EmployeeDep1and2}}$ is equivalent to $\Phi_{\mathsf{EmployeeDep1}} \wedge \Phi_{\mathsf{EmployeeDep2}}$. Since the class EmployeeDep1and2 inherits the integrity constraints of the classes EmployeeDep1 and EmployeeDep2, no additional integrity constraint must be defined on class EmployeeDep1and2.

– The class ManagerDep1 consists of the base extensions 3–5:

$$\Phi_{\mathsf{ManagerDep1}} = 3 \vee 4 \vee 5$$

The integrity constraint set of class ManagerDep1 is equivalent to the conjunction of the inherited integrity constraints.

– The class ManagerDep2 consists only of the base extensions 5–6:

$$\Phi_{\mathsf{ManagerDep2}} = 5 \vee 6$$

Since the conjunction of the inherited integrity constraints is equivalent to the disjunction of the base integrity constraints 4–6, here an additional constraint is required to exclude the base integrity constraint 4.

– Finally, the class ManagerDep1and2 consists only of the base extension 5:

$$\Phi_{\mathsf{ManagerDep1and2}} = 5$$

Since $\Phi_{\mathsf{ManagerDep1and2}}$ is equivalent to the conjunction of the inherited integrity constraints, no additional integrity constraint is needed for this class.

7 Conclusions

In this paper, we worked out the correspondence between integrity constraints and extensional relationships as foundation for semantic schema integration. We pointed out the relevance of considering integrity constraints during schema integration. Exemplary we showed which problems may occur when integrity constraints of local schemas are not integrated. So, we come to the conclusion that, as a rule, schema integration without integrity constraints may result in an integrated schema which is not conform to the modeled real world. The resulting schema is too general and it does not capture the correct semantics of the local classes which are expressed by the existing integrity constraints. Thus, on the global level database states may arise which cannot be represented in

the corresponding local databases. On the other hand, if the integrated schema is more restrictive than the local schemas, i.e. if there are additional global constraints, then local understandability cannot be guaranteed.

In order to avoid both problems, a complete and correct schema integration considering integrity constraints as well as their relationship to extensional assertions is necessary. Here, the main problem is to compute the base integrity constraints for a given set of integrity constraints. In case this is possible in polynomial time, the complexity of entire schema integration process is polynomial. Otherwise, in case of arbitrary complex integrity constraints, a correct and complete schema integration is not feasible in polynomial time (see satisfiability or implication problem). Therefore, the input integrity constraint set must be restricted to certain patterns which can be handled in polynomial time, e.g. equality or dense linear order constraints. Nevertheless, if the input integrity constraints are maximal overlapping, then the number of base integrity constraints is exponential. Fortunately, in large schemas only a few classes (i.e. integrity constraint sets) are overlapping. Most of them are disjoint. Thus, it is often possible to derive a (relatively) small set of base integrity constraints.

Acknowledgments: Thanks to K. Schwarz for helpful remarks. This work was partly supported by the German Federal State Sachsen-Anhalt under FKZ 1987/2527R.

References

1. R. M. Alzahrani, M. A. Qutaishat, N. J. Fiddian, and W. A. Gray. Integrity Merging in an Object-Oriented Federated Database Environment. In C. Goble and J. Keane, eds., *Proc. BNCOD-13*, LNCS 940, pp. 226–248. Springer, 1995. 32
2. C. Batini, M. Lenzerini, and S. B. Navathe. A Comparative Analysis of Methodologies for Database Schema Integration. *ACM Computing Surveys*, 18(4):323–364, 1986. 31, 32, 40
3. S. Conrad, I. Schmitt, and C. Türker. Considering Integrity Constraints During Federated Database Design. In S. M. Embury, N. J. Fiddian, A. W. Gray, and A. C. Jones, eds., *Proc. BNCOD-16*, LNCS 1405, pp. 119–133. Springer, 1998. 40
4. M. Garcia-Solaco, M. Castellanos, and F. Saltor. A Semantic-Discriminated Approach to Integration in Federated Databases. In S. Laufmann, S. Spaccapietra, and T. Yokoi, eds., *Proc. CoopIS'95*, pp. 19–31, 1995. 32, 34
5. M. Garcia-Solaco, F. Saltor, and M. Castellanos. Semantic Heterogeneity in Multidatabase Systems. In O. A. Bukhres and A. K. Elmagarmid, eds., *Object-Oriented Multidatabase Systems*, pp. 129–202, Prentice Hall, 1996. 31
6. R. Hull. Relative Information Capacity of Simple Relational Database Schemata. *SIAM Journal on Computing*, 15(3):856–886, 1986. 32
7. J. A. Larson, S. B. Navathe, and R. Elmasri. A Theory of Attribute Equivalence in Databases with Application to Schema Integration. *IEEE Transactions on Software Engineering*, 15(4):449–463, 1989. 31
8. M. V. Mannino, B. N. Navathe, and W. Effelsberg. A Rule-based Approach for Merging Generalization Hierarchies. *Information Systems*, 13(3):257–272, 1988. 32, 34
9. A. Motro. Superviews: Virtual Integration of Multiple Databases. *IEEE Transactions on Software Engineering*, 13(7):785–798, 1987. 31
10. S. B. Navathe, R. Elmasri, and J. A. Larson. Integrating User Views in Database Design. *IEEE Computer*, 19(1):50–62, 1986. 32, 34

11. M. P. Reddy, B. E. Prasad, and A. Gupta. Formulating Global Integrity Constraints during Derivation of Global Schema. *Data & Knowledge Engineering*, 16(3):241–268, 1995. 32
12. I. Schmitt and G. Saake. Merging Inheritance Hierarchies for Database Integration. In M. Halper, ed., *Proc. CoopIS'98*, pp. 322–331. IEEE CS Press, 1998. 40, 42
13. A. P. Sheth and J. A. Larson. Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases. *ACM Computing Surveys*, 22(3):183–236, 1990. 31
14. S. Spaccapietra, C. Parent, and Y. Dupont. Model Independent Assertions for Integration of Heterogeneous Schemas. *The VLDB Journal*, 1(1):81–126, 1992. 32, 34
15. C. Türker and G. Saake. Deriving Relationships between Integrity Constraints for Schema Comparison. In W. Litwin, T. Morzy, and G. Vossen, eds., *Proc. ADBIS'98*, LNCS 1475, pp. 188–199. Springer, 1998. 35
16. M. W. W. Vermeer and P. M. G. Apers. The Role of Integrity Constraints in Database Interoperation. In T. M. Vijayaraman, A. P. Buchmann, C. Mohan, and N. L. Sarda, eds., *Proc. VLDB'96*, pp. 425–435. Morgan Kaufmann, 1996. 32

Deriving Type Conflicts and Object Cluster Similarities in Database Schemes by an Automatic and Semantic Approach

Domenico Ursino

Dipartimento di Elettronica, Informatica e Sistemistica
Università della Calabria
Via Pietro Bucci, 87036 Rende (CS), Italy
ursino@si.deis.unical.it

Abstract. This paper proposes an automatic, probabilistic approach to the detection of type conflicts and object cluster similarities in database schemes. The type of an object indicates if it is an entity, a relationship or an attribute; type conflicts indicate the existence of objects representing the same concept yet having different types. Object cluster similarities denote similitudes between portions of different schemes. The method we are proposing here is based on considering pairs of objects having different types (resp. pairs of clusters), belonging to different schemes and on measuring their similarity. To this purpose object (resp. cluster) structures as well as object (resp. cluster) neighborhoods are analyzed to verify similitudes and differences.

1 Introduction

In the last years, an enormous increase of data available in electronic form has been witnessed, as well as a corresponding proliferation of query languages, data models and systems for managing these data. Nowadays, heterogeneous data management systems often coexist within the same operational environment. Traditional approaches to data management do not seem to guarantee, in these cases, the needed level of access transparency to stored data while preserving the autonomy of local databases. Cooperative Information Systems have been proposed in recent years as a solution to these problems.

Cooperative Information System architectures (such as the mediator-wrapper one [15,13]) are based on a global dictionary storing properties between objects belonging to different schemes (*interscheme properties*); these are fundamental for reconciliating heterogeneities and providing to the end user a uniform access to various databases. Interscheme properties can be classified as:

- *Nominal*: these are synonymies (i.e. two objects have different names but the same meaning and the same type), homonymies (i.e. two objects have the same name and the same type but different meanings) and hypernymies (i.e. one object has a more general meaning than another one of the same type).
- *Structural*: these are essentially inclusions of an object into another one.

Eder et al. (Eds.): ADBIS'99, LNCS 1691, pp. 46–60, 1999.
© Springer-Verlag Berlin Heidelberg 1999

- *Type Conflicts*: the type of an object O indicates if O is an entity, a relationship or an attribute[1]; therefore there exists a type conflict between two objects if they represent the same concept, yet having different types. The existence of a type conflict between two objects denotes that they are two different representations of the same portion of the reality; such conflicts derive from the fact that different designers may have different perceptions of the same reality.
- *Object Cluster Similarities*: an object cluster represents a set of connected objects in a scheme, i.e., a subscheme; an object cluster similarity denotes the similitude between two portions of two different schemes.

In the literature a certain number of approaches for discovering nominal and structural properties has been proposed [2,3,4,5,6,8,9,12]; on the contrary, there is a little number of proposed approaches for discovering type conflicts and object cluster similarities [8,10,12].

When the amount of involved data is huge and/or when they change quite frequently over time, manual approaches to derive interscheme properties do not suffice and the necessity arises of automatic techniques.

In this paper we propose a novel automatic technique for detecting type conflicts and object cluster similarities in database schemes defined by E/R diagrams. The approach is based on computing a similarity degree between pairs of object $O_1 \in S_1$ and $O_2 \in S_2$ having different types (for type conflict detection) or pairs of object clusters $C_1 \in S_1$ and $C_2 \in S_2$ (for object cluster similarity detection). Similarity degrees are expressed using values in the interval [0..1]. Type conflicts and object cluster similarities are detected in the first place as probabilistic properties of scheme objects. Then, only those characterized by a sufficiently high similarity degree are returned as actual detected conflicts.

In order to derive similarity degrees associated to type conflicts (object cluster similarities, resp.), for each scheme object (cluster, resp.) our algorithm considers its structure and its context [5,9]. The *structure* of an object consists of its attributes whereas the structure of a cluster consists of the objects belonging to it. The *context* of an object O (resp., of a cluster C) indicates the objects belonging to the neighborhood (see below) of O (resp., of C).

Most of the automatic approaches proposed in the literature for deriving interscheme properties are based on comparing structures; only few of them consider also object contexts. Our approach exploits structures and contexts in deriving similarity degree because we argue that *(i)* objects having the same real world semantics are very often characterized by the presence of common elements in their context; *(ii)* it is generally accepted that similarity techniques derived taking into account contexts are more precise than techniques based only on the attribute analysis [4,5,9]. Finally, for attributes we have to take into account their *semantic relevance* [6], i.e. their relevance in distinguishing the semantics of entities/relationships they belong to.

The proposed methodology for the extraction of type conflicts and object cluster similarities exploits some basic background knowledge about involved databases consisting in:

[1] Scheme object types are sometimes called meta-types in the literature.

- A dictionary of Lexicographic Synonymy Properties $LSPD$, storing triplets of the form $[A, B, f]$, where A and B are involved objects and f is a fuzzy coefficient expressing the strength of the property. We assume that such synonymies are derived from a standard thesaurus; however, further synonymies, more specific to the application domain, can be also provided by DBAs.
- A Synonymy Property Dictionary SPD, storing tuples of the form $\lfloor A, B, f \rfloor$, where A and B are involved objects and f is a fuzzy coefficient that expresses the strength of the property. In the literature some algorithms have been proposed for extracting synonymy properties and for constructing a Synonymy Dictionary [4,5,6,8,9,12].

The approach presented in this paper uses some weights and thresholds; we have exploited several example cases for tuning up their optimal values and for validating the approach itself after the tuning phase. The most interesting and complete test case has been the set of database schemes relative to Italian Central Governmental Offices (ICGO). ICGO own about 300 databases many of which having a complex structure and containing large amounts of data. First a tuning phase has been carried out for fixing weights and thresholds; then the algorithms have been applied and their results have been validated. To this purpose we have compared results produced by our approach with those obtained manually by "Italian Information Systems Authority for Public Administration" (AIPA); the comparison has demonstrated that our algorithms are capable of producing high quality results yet requiring a short amount of time.

We point out that the approach described in this paper for type conflicts and object cluster similarities follows the same philosophy of [9]. Therefore, in the whole, we propose a unified semi-automatic approach for deriving nominal properties, type conflicts and object cluster similarities. To the best of our knowledge, in the literature, there is no other uniform approach for deriving all these kinds of interscheme properties.

The plan of the paper is as follows. In the next section we describe the type conflict detection whereas the derivation of object cluster similarities is illustrated in Section 3. Related works are finally considered in Section 4.

2 An algorithm for detecting type conflicts

2.1 Preliminaries

In the following we suppose that all schemes under consideration are represented using the E/R model described in [1]. In addition the following concepts will be exploited:

Definition 1. Let A be an *attribute*. The *structure* of A consists of A itself. The *context* of A consists both of the object O, which A belongs to, and of the structure of O (except, obviously, A itself).

Let E be an *entity*. The *structure* of E consists of the set of its attributes. The *context* of E consists of relationships it takes part into along with their

attributes and all other entities linked to these relationships along with their attributes.

Let R be a *relationship*. The *internal structure* of R consists of its attributes. The *structure* of R consists of its internal structure plus the entities linked by R and their attributes. The *context* of R consists of entities linked (through some relationship) to entities belonging to the structure of R. □

In the sequel, we shall use the following support functions: $EStructure(E)$ (resp., $RStructure(R)$), that takes in input an entity E (resp., a relationship R) and yields in output objects belonging to its structure; $AContext(A)$ (resp., *(i)* $EContext(E)$, *(ii)* $RContext(R)$) that receives an attribute A (resp., *(i)* an entity E, *(ii)* a relationship R) and returns the set of objects belonging to its context.

2.2 General characteristics

In this section we illustrate an algorithm for the derivation of type conflicts. Each type conflict will be denoted by a triplet of the form $\lceil A, B, f \rceil$, where A and B are the involved objects and f is a fuzzy coefficient, in the real interval $[0, 1]$, which expresses the strength of the property.

Our method consists of several new algorithms for deriving type conflicts between objects. For each object, we consider both its structure (taking also into account the semantic relevance of attributes forming it) and its context. The main algorithm for extracting type conflicts is as follows:

Algorithm for extracting type conflicts
Input: a list S of n database schemes; a dictionary SPD of synonymy properties and a dictionary $LSPD$ of lexicographic synonymy properties;
Output: a dictionary TCD of type conflicts;
begin
$TCD := EA\_E\_Conf(S, SPD, LSPD) \cup RA\_E\_Conf(S, SPD, LSPD) \cup$
$EA\_R\_Conf(S, SPD, LSPD) \cup RA\_R\_Conf(S, SPD, LSPD) \cup$
$E\_R\_Conf(S, SPD, LSPD)$
end

The algorithm constructs the dictionary TCD by activating some functions, one for each kind of type conflicts. EA_E_Conf derives conflicts between an entity attribute and an entity; RA_E_Conf derives conflicts between a relationship attribute and an entity; EA_R_Conf detects conflicts between an entity attribute and a relationship; RA_R_Conf determines conflicts between a relationship attribute and a relationship; finally E_R_Conf detects conflicts between an entity and a relationship. In the following, due to space limitations, we show in details only the behaviour of EA_E_Conf.

2.3 Discovering conflicts between an entity attribute and an entity

The function *EA_E_Conf* derives conflicts between an entity attribute and an entity. This derivation is obtained into two separate phases: the first phase determines rough type conflicts, taking into account only object structures, whereas

the second phase extracts refined type conflicts by using both rough type conflicts and object contexts. In more details the function is the following:

Function *EA_E_Conf* (S: a list of n database schemes; SPD: a Synonymy Property Dictionary; $LSPD$: a Lexicographic Synonymy Property Dictionary):
return a set of type conflicts between an entity attribute and an entity;
var
  RTC: a set of (rough) type conflicts between an entity attribute and an entity;
  TTC: a set of type conflicts between an entity attribute and an entity;
begin
  RTC := *EA_E_Rough(S,LSPD)*;
  TTC := *EA_E_Refined(S,SPD,LSPD,RTC)*;
  TTC := *EA_E_Discard_Weak(TTC)*;
  return TTC
end

Computing rough entity attribute - entity conflicts The function for computing rough conflicts between entity attributes and entities analyses pairs of distinct schemes. For each pair of objects in the two schemes, and representing an entity attribute and an entity, resp., the function computes a coefficient denoting the strength of the similarity of the objects of the pair. To this end, it considers similarities of all possible pairs of attributes, the former being the entity attribute under examination and the latter being one of the attributes of the entity we are considering. The corresponding function is as follows:

Function *EA_E_Rough* (S: a list of n schemes; $LSPD$: a Lexicographic Synonymy Property Dictionary): **return** a set of (rough) type conflicts between an entity attribute and an entity;
var
  SP: a set of synonymies between objects;
  TTC: a set of type conflicts between an entity attribute and an entity;
  h, k : 1 .. n; f: set of reals in the interval [0..1];
begin
  $TTC := \emptyset$;
  for h:= 1 **to** n **do**
    for k:= 1 **to** n **do**
      for each pair consisting of an entity attribute $A_i \in S_h$ and an entity $E_j \in S_k$ **do begin**
        $SP := \emptyset$;
        for each attribute $A_m \in EStructure(E_j)$ **do**
          $SP := SP \cup \langle A_i, A_m, W\_Mean(A_i, A_m, LSPD)\rangle$;
        $SP := SP \cup \langle A_i, E_j, Lex\_Syn(A_i, E_j, LSPD)\rangle$;
        $f_{A_i E_j} := Max(SP)$;
        $TTC := TTC \cup \lceil A_i, E_j, f_{A_i E_j} \rceil$
      end;
  return TTC
end

The function *W_Mean* computes the synonymy coefficient associated to a pair of attributes (A_l, A_m) taken in input. To this end, three characteristics are considered, namely, attribute name, domain, and key characterization; the key characterization of an attribute tells if the attribute is a primary key (PK), a secondary key (SK) or not a key (NK) for the entity it belongs to. The function returns the result of the expression $W_N \times N_{A_l A_m} + W_D \times D_{A_l A_m} + W_K \times K_{A_l A_m}$, where W_N, W_D and W_K are weighting factors, whose optimal values have been experimentally set to $0.8, 0.15$ and 0.05, respectively; $N_{A_l A_m}$ is the value of the lexicographic synonymy between A_l and A_m, as resulting from the dictionary $LSPD$; the value of $D_{A_l A_m}$ is set to 1 if the domains of A_l and A_m are the same, to 0.5 if they are compatible (default compatibilities are integers with reals and chars with strings; further, more specific, compatibilities can be provided by the DBA) and to 0 otherwise. Finally, $K_{A_l A_m}$ is set to 1 if A_l and A_m are both PK or both SK or both NK, to 0.5 if one of them is PK and the other is SK or if one of them is SK and the other is NK, to 0 if one of them is PK and the other is NK.

The function *Lex_Syn* checks, in the $LSPD$, the presence of a lexicographic synonymy between names of an attribute A_i and an entity E_j, which it receives in input, and yields in output its plausibility coefficient, if it exists, 0 otherwise.

The function *Max* takes in input the set SP of synonymies between objects and yields in output the maximum of all plausibility values.

Computing refined entity attribute - entity conflicts This function derives more refined conflicts between an entity attribute and an entity by exploiting: *(i)* rough type conflicts derived by the previous procedure, *(ii)* synonymies between attributes of the context, *(iii)* synonymies between entities of the context. The function is as follows:

Function *EA_E_Refined*(S: a list of n schemes; SPD: a Synonymy Property Dictionary; $LSPD$: a Lexicographic Synonymy Property Dictionary; RTC: a set of (rough) type conflicts between an entity attribute and an entity): **return** a set of type conflicts between an entity attribute and an entity;
var
  A, Q: set of reals in the interval [0..1]; h, k : $1..n$; g: Real;
  TTC: a set of type conflicts between an entity attribute and an entity;
begin
  $TTC := \emptyset$;
  for $h := 1$ **to** n **do**
    for $k := 1$ **to** n **do**
      for each pair consisting of an entity attribute $A_i \in S_h$ and an entity $E_j \in S_k$ **do begin**
        $A_{A_i E_j} := EA\_E\_Att\_Syn(A_i, E_j, LSPD)$;
        $Q_{A_i E_j} := EA\_E\_Ent\_Syn(A_i, E_j, SPD)$;
        Let $\lceil A_i, E_j, f_{A_i E_j} \rceil$ a tuple belonging to RTC;
        $g := EA\_E\_Refined\_Val(A_{A_i E_j}, Q_{A_i E_j}, f_{A_i E_j})$;
        $TTC := TTC \cup \lceil A_i, E_j, g \rceil$
      end;
  return TTC
end

The function *EA_E_Att_Syn* computes synonymies between attributes belonging to the context of an attribute A_i and the context of an entity E_j, which it receives in input. This function is as follows:

Function *EA_E_Att_Syn*(A_i: an entity attribute; E_j: an entity; $LSPD$: a Lexicographic Synonymy Property Dictionary): **return** a Real $\in [0..1]$;
var
  SP: a set of synonymies between attributes;
  g: Real $\in [0, 1]$; L, M : a set of attributes; F: a matrix of reals;
begin
  $SP := \emptyset$;
  $L := \{\ A_l \mid A_l$ is an attribute *and* $A_l \in AContext(A_i)\ \}$;
  $M := \{\ A_m \mid A_m$ is an attribute *and* $A_m \in EContext(E_j)\ \}$;
  for each pair of attributes $A_l \in L$ and $A_m \in M$ **do**
    $SP := SP \cup \langle A_l, A_m, W\_Mean(A_l, A_m, LSPD)\rangle$;
  $SP := Obj\_Discard(SP)$;
  $F := Mat(SP)$;
  $g := Matching(L, M, F, \omega_v)$;
  return g
end

The function *W_Mean* is the same as that defined in *EA_E_Rough*. The function *Obj_Discard* considers each tuple belonging to SP and normalizes the corresponding plausibility value, by setting it to 0 if this was previously under a certain given threshold th_v. The optimal value of th_v has been experimentally set to 0.25. The function $Mat(SP)$ creates a matrix F having a row for each $A_l \in L$, a column for each $A_m \in M$ and $F[A_l, A_m]$ equal to the plausibility value associated in SP to the pair A_l and A_m.

The function *Matching* computes a factor obtained from calculating a maximum weight matching. The input here are two sets of objects $L = \{l_1, \ldots, l_r\}$ and $M = \{m_1, \ldots, m_s\}$ and a cost matrix F on L and M such that, for each $l_i \in L$ and $m_j \in M$, $0.0 \leq F[l_i, m_j] \leq 1.0$. The output is a value v in the real interval [0..1]. If $L = \emptyset$ or $M = \emptyset$ then *Matching* returns $v = 0$. Otherwise, let $BG = (L \cup M, A)$ be a bipartite weighted graph, where A is the set of weighted edges $\{(l_i, m_j, f_{ij}) \mid f_{ij} > 0\}$; the maximum weight matching for BG is a set $A' \subseteq A$ of edges such that for each node $x \in L \cup M$ there is at most one edge of A' incident onto x and $\phi(A') = \sum_{(l_i, m_j, f_{ij}) \in A'} f_{ij}$ is maximum[2]. Now, let $\overline{\phi}(A') = \frac{\phi(A')}{|A'|}$. The value v returned by *Matching* is defined as:

$$v = 1 - \tfrac{1}{2} \times \omega_v \tfrac{abs(|L|-|M|)+2\times(min(|L|,|M|)-|A'|)}{|L|+|M|} \times \overline{\phi}(A')$$

where ω_v is used to weight the importance of unrelated edges. The optimal value of ω_v has been experimentally set to 1.

The function *EA_E_Ent_Syn* takes in input an attribute A_i, an entity E_j and a dictionary SPD of synonymy properties and returns a synonymy coefficient as the maximum of plausibility coefficients relative to all entities related to the entity E_j and the entity which A_i belongs to.

[2] A survey on maximum weight matching can be found in [7].

The function *EA_E_Refined_Val* takes in input the real numbers $A_{A_iE_j}$, $Q_{A_iE_j}$ and $f_{A_iE_j}$ and computes their weighted mean with the coefficients $V_A = 0.3$, $V_Q = 0.25$ and $V_f = 0.45$.

Discarding weak type conflicts *EA_E_Discard_Weak* takes in input a set TTC of (candidate) type conflicts between an entity attribute and an entity. The function computes an interest threshold: $th_{ae} = max\left(\delta, \frac{TTC_{MAX}+TTC_{MIN}}{2}\right)$, where TTC_{MAX} and TTC_{MIN} are the maximum and the minimum plausibility values in TTC and δ is a real number $\in [0..1]$ (its value has been experimentally set to 0.6). The function then discards all type conflicts whose coefficients are smaller than th_{ae}.

As already stated, all type conflicts not discarded by *EA_E_Discard_Weak* are added to the Type Conflict Dictionary TCD.

2.4 Discovering other type conflicts

Discovering *conflicts between a relationship attribute and an entity* is carried out by the function RA_E_Conf; this is very similar to EA_E_Conf, the only relevant difference is that it considers the structure and the context of a relationship attribute instead of the structure and the context of an entity attribute.

Discovering *conflicts between an attribute and a relationship* is carried out by functions EA_R_Conf and RA_R_Conf; these are analogous to EA_E_Conf and RA_E_Conf, the only relevant difference is that it considers the structure and the context of a relationship, instead of the structure and the context of an entity.

Discovering *conflicts between an entity and a relationship* is realized by the function E_R_Conf; it follows an approach similar to EA_E_Conf except that the structure and the context of a relationship is considered instead of the structure and the context of an entity attribute.

2.5 An example taken from the literature

Consider the schemes represented in Figure 1 (example taken from [12]):

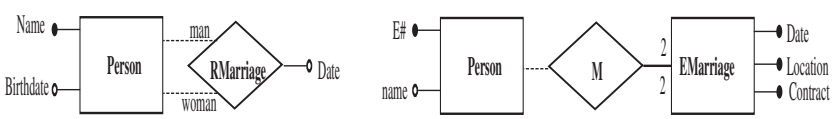


Fig. 1. Two different representations of marriages

Assume the $LSPD$ contains the tuples $[Date, Date, 1]$ and $[Date, BirthDate, 0.9]$. By applying the procedure E_R_Conf to these schemes we derive a type conflict between $EMarriage$ and $RMarriage$ with a plausibility factor $f = 0.66$. The interested reader can find all details of this computation in [14].

3 Extraction of similarities among object clusters

3.1 Preliminaries

There are many situations where a certain portion of the world representation is incorporated as two different *sets of objects* (hereafter called *object clusters*) within two schemes. In order to single out such structural similarities we have designed a second group of algorithms which are presented in this section. Preliminarily consider the following definitions:

Definition 2. An *object cluster* is either *(i)* a relationship cluster, or *(ii)* a subset cluster, or *(iii)* a composed cluster. A *relationship cluster* is a portion of an E-R scheme consisting of a relationship, the entities it links and all involved attributes. A *subset cluster* is a portion of an E-R scheme consisting of an entity, the entities linked to it by an *is-a* relationships and all involved attributes. A *composed cluster* is a cluster including in turn two object clusters sharing at least one entity. □

Definition 3. Given *a cluster* C, the *structure of* C consists of objects (attributes, entities and relationships) belonging to C; the *context of* C consists of the difference between the union of the cluster object contexts and objects belonging to the cluster itself (and therefore belonging to the cluster's structure). □

In the sequel, we shall use the functions $CStructure(C)$ and $CContext(C)$ which takes in input a cluster C and yields in output objects belonging to its structure and to its context, resp.

For each cluster, our algorithm takes into account both the structure and the context. The algorithm for extracting object cluster similarities is as follows:

Algorithm for extracting object cluster similarities
Input: a list S of n database schemes; a dictionary SPD of synonymy properties; a dictionary $LSPD$ of lexicographic synonymy properties;
Output: a dictionary $OCSD$ of similarities between object clusters;
var
$ROCS$: a set of (rough) similarities between object clusters;
$OCPS$: a set of (interesting) pairs of object clusters;
begin
$OCPS := Select\_Interesting\_Clusters(S, SPD)$;
$ROCS := OC\_Rough(S, OCPS, SPD, LSPD)$;
$OCSD := OC\_Refined(S, OCPS, SPD, ROCS)$;
$OCSD := OC\_Discard\_Weak(OCSD)$;
return $OCSD$
end

3.2 Selecting interesting clusters

The function for selecting interesting clusters is devoted to limit the (potentially exponential) number of possible clusters by determining the most interesting, i.e. the most promising ones.

An object cluster is to be considered as the minimum subscheme containing two "seed" entities, called *seed_ent1* and *seed_ent2*. The term "minimum subscheme" indicates the subscheme containing the minimum number of objects, among those comprising both *seed_ent1* and *seed_ent2*. If more than one subscheme exists with the same (minimum) number of objects, that having the maximum number of objects involved in synonymies stored in the SPD is chosen. If more than one exists with this characteristic, that having the greatest plausibility coefficients, associated to the synonymies which it is involved in, is taken.

For each pair of schemes S_h and S_k belonging to S, the function determines all pairs of object clusters (C_i, C_j) such that $C_i \in S_h$, $C_j \in S_k$ and both C_i and C_j have been judged promising. More in particular, in a first step, object clusters of S_h are those having, as *seed_ent1*, an entity E_r involved in a significant entity synonymy and, as *seed_ent2*, any other entity E_l of S_h; given an object cluster in S_h having E_r and E_l as seeds, corresponding object clusters in S_k are those having, as *seed_ent1*, any entity E_s related to E_r by a significant synonymy and, as *seed_ent2*, any entity E_m related to E_l by a significant synonymy. The second step is dual to the first one. The function can be encoded as follows:

Function *Select_Interesting_Clusters*(S: a list of n database schemes; SPD: a Synonymy Property Dictionary): **return** a set of (interesting) pairs of object clusters;
var
  $OCPS$: a set of (interesting) pairs of object clusters;
begin
  for each $\lfloor E_r, E_s, f_{E_r E_s} \rfloor \in SPD$ **do begin**
    Let S_h the scheme of E_r and S_k the scheme of E_s;
    for each entity $E_l \in S_h$ **do**
      for each $\lfloor E_l, E_m, f_{E_l E_m} \rfloor \in SPD$ **such that** $E_m \in S_k$ **do begin**
        $C_i := Minimum\_Cluster(E_r, E_l)$;
        $C_j := Minimum\_Cluster(E_s, E_m)$;
        $OCPS := OCPS \cup (C_i, C_j)$
      end;
    for each entity $E_p \in S_k$ **do**
      for each $\lfloor E_p, E_q, f_{E_p E_q} \rfloor \in SPD$ **such that** $E_q \in S_h$ **do begin**
        $C_i := Minimum\_Cluster(E_r, E_q)$;
        $C_j := Minimum\_Cluster(E_s, E_p)$;
        $OCPS := OCPS \cup (C_i, C_j)$
      end
  end
end

The function $Minimum\_Cluster$ takes in input two seed entities E_i and E_j and yields in output the object cluster (i.e. the "minimum subscheme") which E_i and E_j belong to.

3.3 Extraction of rough synonymies between object clusters

The function for computing rough object cluster similarities determines, for each pair of object clusters, a coefficient stating the associated similarity degree; this is a weighted mean of similarity degrees of attributes, entities, relationships and cardinalities relative to clusters of the pair. The function is as follows:

Function *OC_Rough*(S: a list of n database schemes; $OCPS$: a set of (interesting) object cluster pairs; SPD: a Synonymy Property Dictionary; $LSPD$: a Lexicographic Synonymy Property Dictionary): **return** a set of (rough) object cluster similarities;
var
  Q, T, U, V: a set of reals $\in [0..1]$; f: Real;
  $TOCS$: a set of similarities between object clusters;
begin
  $TOCS := \emptyset$;
  for each pair of object clusters $(C_i, C_j) \in OCPS$ **do begin**
    $Q_{C_iC_j} := OC\_Rough\_Att\_Syn(C_i, C_j, LSPD)$;
    $T_{C_iC_j} := OC\_Rough\_Ent\_Syn(C_i, C_j, SPD)$;
    $U_{C_iC_j} := OC\_Rough\_Rel\_Syn(C_i, C_j, SPD)$;
    $V_{C_iC_j} := OC\_Rough\_Card\_Syn(S, C_i, C_j)$;
    $f := OC\_Rough\_Val(Q_{C_iC_j}, T_{C_iC_j}, U_{C_iC_j}, V_{C_iC_j})$;
    $TOCS := TOCS \cup \|C_i, C_j, f\|$
  end;
  return $TOCS$
end

Functions *OC_Rough_Att_Syn*, *OC_Rough_Ent_Syn* and *OC_Rough_Rel_Syn* are analogous to the function *EA_E_Att_Syn*, the only differences being that:

- in *OC_Rough_Att_Syn*:
  - $L := \{A_l \mid A_l$ is an attribute *and* $A_l \in CStructure(C_i)\}$
  - $M := \{A_m \mid A_m$ is an attribute *and* $A_m \in CStructure(C_j)\}$.
- in *OC_Rough_Ent_Syn*:
  - $L := \{E_l \mid E_l$ is an entity *and* $E_l \in CStructure(C_i)\}$
  - $M := \{E_m \mid E_m$ is an entity *and* $E_m \in CStructure(C_j)\}$,
  - $SP := \bigcup_{\lfloor E_l, E_m, f_{E_lE_m} \rfloor \in SPD, E_l \in L, E_m \in M} \langle E_l, E_m, f_{E_lE_m} \rangle$.
- in *OC_Rough_Rel_Syn*:
  - $L := \{R_l \mid R_l$ is a relationship *and* $R_l \in CStructure(C_i)\}$;
  - $M := \{R_m \mid R_m$ is a relationship *and* $R_m \in CStructure(C_j)\}$,
  - $SP := \bigcup_{\lfloor R_l, R_m, f_{R_lR_m} \rfloor \in SPD, R_l \in L, R_m \in M} \langle R_l, R_m, f_{R_lR_m} \rangle$

The function *OC_Rough_Card_Syn* takes in input a list S of schemes and two object clusters C_i and C_j; it considers cardinalities of relationships constituting the structure of clusters and returns a real value $\in [0..1]$; the higher the number of relationships of C_i, having the same cardinalities of the corresponding synonym relationships of C_j, is, the higher the value returned in output is. A simple

implementation of the function should return the ratio between the number of synonym relationships having the same cardinality and the total number of synonym relationships.

The function *OC_Rough_Val* takes in input four real coefficients: $Q_{C_iC_j}$, $T_{C_iC_j}$, $U_{C_iC_j}$ and $V_{C_iC_j}$ and computes their weighted mean with the coefficients $Z_Q = 0.5$, $Z_T = 0.3$, $Z_U = 0.15$ and $Z_V = 0.05$.

3.4 Extraction of refined similarities between object clusters and discard of weak ones

The function for computing refined object cluster similarities is devoted to derive more refined cluster similarities. To this purpose it exploits entity and relationship synonymies, stored in the SPD, and rough object cluster similarities, stored in the $ROCS$. More in details the function is as follows:

Function *OC_Refined*(S: a list of n database schemes; $OCPS$: a set of (interesting) object cluster pairs; SPD: a Synonymy Property Dictionary; $ROCS$: a set of (rough) object cluster similarities): **return** a set of object cluster similarities;
var
  T, U: set of reals $\in$ [0..1]; g: Real;
  $TOCS$: a set of similarities between object clusters;
begin
  $TOCS := \emptyset$;
  for each pair of object clusters $(C_i, C_j) \in OCPS$ **do begin**
    $T_{C_iC_j} := OC\_Refined\_Ent\_Syn(C_i, C_j, SPD)$;
    $U_{C_iC_j} := OC\_Refined\_Rel\_Syn(C_i, C_j, SPD)$;
    Let $\lfloor C_i, C_j, f_{C_iC_j} \rfloor$ a tuple belonging to $ROCS$;
    $g := OC\_Refined\_Val(f_{C_iC_j}, T_{C_iC_j}, U_{C_iC_j})$;
    $TOCS := TOCS \cup \|C_i, C_j, g\|$
  end;
  return $TOCS$
end

The function *OC_Refined_Ent_Syn* is analogous to the function *OC_Rough_Ent_Syn*, the only differences being that:

- $L := \{E_l \mid E_l$ is an entity *and* $E_l \in CContext(C_i)\}$
- $M := \{E_m \mid E_m$ is an entity *and* $E_m \in CContext(C_j)\}$.

The function *OC_Refined_Rel_Syn* is analogous to the function *OC_Rough_Rel_Syn*, the only differences being that:

- $L := \{R_l \mid R_l$ is a relationship *and* $R_l \in CContext(C_i)\}$
- $M := \{R_m \mid R_m$ is a relationship *and* $R_m \in CContext(C_j)\}$.

The function *OC_Refined_Val* takes in input three coefficients $f_{C_iC_j}$, $T_{C_iC_j}$ and $U_{C_iC_j}$ and computes their weighted mean with the coefficients $Z_f = 0.65$, $Z_T = 0.25$ and $Z_V = 0.1$.

The function *OC_Discard_Weak* is similar to the function *EA_E_Discard_Weak* except that now the threshold th_{oc} is equal to $max\left(\sigma, \frac{OCSD_{MAX}+OCSD_{MIN}}{2}\right)$ (σ has been experimentally set to 0.55) and that properties judged interesting are stored in the final Object Cluster Similarity Dictionary.

3.5 An example taken from the literature

Consider the schemes in Figure 2, taken from [12]. Consider the cluster $C_1 = (Customer - Ordered - Product)$ and $C_2 = (Customer - Places - Order - Ordline - Product)$. Suppose that the $LSPD$ stores the tuples $[Name, Name, 1]$, $[Date, ODate, 0.85]$, $[Quantity, Qty, 0.95]$, $[P\#, P\#, 1]$. By applying the approach presented in [9] the following synonymies can be found and stored in the SPD: $\lfloor Customer, Customer, 0.72 \rfloor$, $\lfloor Product, Product, 0.75 \rfloor$, $\lfloor Ordered, Ordline, 0.60 \rfloor$, $\lfloor Ordered, Places, 0.47 \rfloor$.

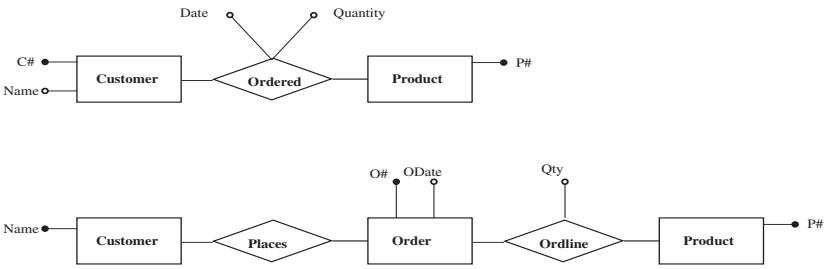


Fig. 2. Two representations of the same world

The application of the algorithm for extracting object cluster similarities detects the presence of a similarity between C_1 and C_2 with a plausibility factor $f = 0.83$. All details of this computation, as well as a dissertation about the role of the context in this derivation, can be found in [14].

4 Related works

In the literature there are very few papers dealing with the detection of type conflicts (e.g., [2,12,8]). In [2] a technique is proposed to solve all kinds of type conflicts, but this paper supposes type conflicts have been detected manually by DBAs and, therefore, does not propose any approach for detecting them.

On the contrary [12] proposes a techniques for detecting type conflicts and object cluster similarities in a semi-automatic fashion; as illustrated also by various examples, techniques presented here are able to detect all kinds of type conflicts derived by techniques described in [12], but in a more automatic fashion. In addition, as previously pointed out, we have designed a unique framework for deriving synonymies, homonymies, type conflicts and object cluster similarities; the techniques presented in [12] consider only type conflicts and object cluster similarities.

Our approach has some similarities with that presented in [8] but: *(i)* [8] derives only nominal properties and type conflicts (and not object cluster similarities); *(ii)* it requires a heavier intervention of the DBA since he must validate

one "assumed similarity" at each step (whereas our algorithm requires the intervention of the DBA only after the derivation of all properties for validating them); *(iii)* finally, we take advantage of the background knowledge whereas this is a lack of [8], as denoted by the authors themselves.

The approach presented in this paper is complementary to that described in [11]: indeed [11] improves the description of types by associating a semantics to each of them in order to explain the terms of application domain and their interrelations; these improvements on type description could be exploited for refining the part of our approach which detects type conflicts. In addition the approach for semantic interoperability presented in [11] assumes the existence of a shared ontology for schemes into consideration and admits that its construction is a difficult task; the part of our approach detecting terminological properties between scheme objects can provide this ontology.

Acknowledgements The Author thanks L. Palopoli, L. Pontieri, G. Terracina and D. Saccà for many inspiring discussions about the arguments of the paper.

References

1. C. Batini, S. Ceri, S.B.Navathe, *Conceptual Database Design*, The Benjamin/Cummings Publishing Company, 1992. 48
2. C. Batini, M. Lenzerini, A methodology for data schema integration in the entity relationship model, *IEEE TSE* 10(6), 650-664, 1984. 47, 58
3. C. Batini, M. Lenzerini, S.B. Navathe, A comparative analysis of methodologies for database scheme integration, *ACM Computing Surveys*, 15(4), 323-364, 1986. 47
4. M.W.Bright, A.R.Hurson, S.Pakzad, Automated Resolution of Semantic Heterogeneity in Multidatabases, *ACM Transactions on Database Systems* 19(2), 212-253, 1994. 47, 48
5. S. Castano, V. De Antonellis, Semantic Dictionary Design for Database Interoperability, *Proceedings of ICDE'97*, Birmingham, United Kingdom, 1997. 47, 48
6. P. Fankhauser, M. Kracker, E.J.Neuhold, Semantic vs. Structural Resemblance of Classes, *SIGMOD RECORD*, 20(4), 59-63, 1991. 47, 48
7. Z. Galil, Efficient algorithms for finding maximum matching in graphs, *ACM Computing Surveys*, 18, 23-38, 1986. 52
8. W. Gotthard, P.C. Lockermann, A. Neufeld, System-Guided View Integration for Object-Oriented Databases, *IEEE TKDE* 4(1), 1-22, 1992. 47, 48, 58, 59
9. L. Palopoli, D. Saccà, D. Ursino, Automatic Derivation of Terminological Properties from Database Schemes, *Proc. DEXA'98*, 90-99, Wien, Austria, 1998. 47, 48, 58
10. L. Palopoli, D. Saccà, D. Ursino, An Automatic Technique for Detecting Type Conflicts in Database Schemes, *Proc. ACM CIKM'98*, 306-313, Bethesda (Maryland), USA, 1998. 47
11. E. Sciore, M. Siegel, A. Rosenthal, Using semantic values to facilitate interoperability among heterogeneous information systems, *ACM TODS* 19(2), 254-290, 1994. 59
12. S.Spaccapietra, C.Parent, View Integration: A Step Forward in Solving Structural Conflicts, *IEEE TKDE* 6(2), 258-274, 1994. 47, 48, 53, 58

13. J.D. Ullman, Information Integration using logical views, *Proceedings ICDT '97*, Delphi, Greece, 19-40, 1997. 46
14. D. Ursino, Deriving type conflicts and object cluster similarities in database schemes by an automatic and semantic approach (extended version), *Technical Report - DEIS CS Laboratory 99-6* . Available from the author. 53, 58
15. G. Wiederhold, Mediators in the architecture of future information systems, *IEEE Computer*, 25(3), 38-49, 1992. 46

A Model for Querying Annotated Documents

Frédérique Laforest and Anne Tchounikine

Laboratoire d'Ingénierie des Systèmes d'Information
INSA, Bat 401, 20 Avenue A. Einstein, 69621 Villeurbanne Cedex, France
laforest@telecom.insa-lyon.fr,
atchouni@if.insa-lyon.fr

Abstract. Most of Information Retrieval models for documents are intended for SGML-like structured documents. In the context of medical informatics, the Patient Record document needs a looser structuration process as an a priori structure can hardly be defined. Thus we propose an authoring tool that allows to annotate embedded information, i.e. to give them a context, with qualifiers that are stored in a thesaurus rather than in SGML-like DTD. The retrieval process in the Patient Records collection takes into account the flexibility of the qualifying process while reformulating the queries (synonymy, generalization and specialization relationships between qualifiers).

1 Introduction

Recent progress in electronic communication techniques (intranet, web) emphases the use of electronic document as an information vector and work basis. Standardisation of the document representation is then required in order to ensure a correct quality in these exchanges. Some norms have appeared. SGML [3] and XML [13] are the best-known norms at the moment. SGML has been and is still widely used in many domains, while XML starts its carrier in the documents communication field. They both allow to describe document grammars (Document Definition Types, DTD) which provide rules on the use of tags in documents. These tags are inserted in a document content to delimit logical elements of its structure. The most "popular" SGML DTD is HTML, which is the standard for the description of Web documents. XML has been written on the basis of SGML, adding (among all) means to describe hyperlinks. It thus can be used to describe Web documents. XML is often compared to HTML even though HTML tags are fixed whereas XML allows for the *creating* of many DTDs: HTML is a language whereas XML, like SGML, is a meta-language. These languages are a first approach to the definition of documents' semantics. They allow to define composition rules for the writing of documents instances. But, the semantic aspect of tags is limited, as it is reduced to the choice of their names and to the definition of their logical structure. The wide use of these standards has relaunched research on information retrieval in documents bases. Indeed, earlier IR models focused on the content of documents and provided systems based on keywords. Sets of keywords were manually composed or obtained by frequency

Eder et al. (Eds.): ADBIS'99, LNCS 1691, pp. 61–75, 1999.
© Springer-Verlag Berlin Heidelberg 1999

calculus on full-text analysis [4]. The use of logical tags inside documents makes natural and even desirable the use of these tags to enhance documents querying. By this mean, requests on documents are not still based only on statistics, but also on the semantics and logic of tags. As we detail it later, computerized patients' records (PR) management systems will gain in using tagged documents as a structuring policy. But the current proposals for the querying of tagged documents remain unfitted for novice users. The complexity of these languages shows they are dedicated to computer scientists. Section 1 presents related work on querying tagged documents. Section 2 presents the context of our research i.e. the PR specificities, and section 3 outline IR requirement. In section 4, we propose a document production model in which the user does not structure his document, but rather *qualifies* the contained information. Section 5 focuses on our IR model that takes into account the flexibility of the indexing achieved by the qualification process and allows queries on the PR composition.

2 Related Work

As stated earlier, authoring standards like SGML (Standard Generalized Markup Language [3], XML (eXtended Markup Language) [13] now allow to logically structure documents. Thus numerous experiments have focused on the way this structuration could be helpful in the collecting of information in documents. A first approach is to "come back" to a well-known problem translating structured documents into traditional DBMS suited formats. Examples of this approach can be found in [12] for relational DB and [6] for object oriented DB. In this last study, documents are matched into an object oriented schema using tuple and list types in order to reproduce the sequence and inclusion concepts of documentary elements. Queries involving the location of an element (e.g. "get the title of the third paragraph") can be computed thanks to the element index in the composed tuple or list. Let us also cite [1] who proposes to use the OEM data model for describing the documents. OEM is a model that has been developed for mediation purposes between heterogeneous databases. OEM allows to represent data whose structure is self-describing and the LOREL query language allows to query these data. These approaches benefit of powerful ad-hoc query and manipulation languages but require to manipulate two different models.
A second approach consists in creating new manipulation and query languages based on the new models provided by documentary standards. For instance SgmlQL [10], UnQl [5], or XML-QL [7] are SQL-like query languages that are intended to query marked documents. Documents are seen as oriented graphs and the (rather complex) query languages are based on tree inclusion algorithms. Our proposal is related to this second kind of approach. Nevertheless the study of the electronic document "Patient Record" (PR) in the context of medical informatics, leads us to look for the loosest structuration and retrieval model. Indeed, as addressed in the second section of this paper, it is neither possible, nor desirable for users to impose a structuring model for PRs. Yet, IR in a PRs collection is strongly related to the way PRs are composed: on the one hand,

queries often focus on the appearance order of information in the PR because this order represents a medical reasoning ; on the other hand, the information semantics depend on their writing context, i.e. on their place in the document (e.g. a drug name in a prescription and in an allergies-related paragraph).

3 The Medical Context

3.1 The Patient Record as Part of the Medical Information System

Information System in the medical area is usually divided into two parts:

- An Administrative management Information System (AIS) which takes into account the administrative facet of the medical organization ;
- A Medical Information System (MIS) which is interested in all the medical information created inside the organization.

The AIS consists mainly in traditional, alphanumeric databases and is out of our present concern. The MIS, which interests us, contains all the information which is directly interesting for the care of patients. The Patient Record (PR), also known as "Medical Record", has been defined as follows:*"The medical record is the memory in which all the data necessary for the surveillance and to take in charge the patient are stored"* [11]. The PR does not only contain the observations of the doctor or the nurses' remarks. It also includes all that can be stored concerning a patient, from demographic data to electro-physiological captures or sophisticated images. The PR thus is the main tool for the centralization and the coordination of the medical activity. It can be used in 2 ways: an individual use to follow the patient at the care unit's level, it is then considered as the history of the medical past of the patient ; and a collective use for epidemiological studies, clinical research... Here, the PR represents a tool for the study and the analysis of public health and is usually used without identification of patients. The PR has few very special characteristics.The PR is multi-author while created, and multi-user while accessed. Indeed he PR is filled by various persons from the medical staff: physicians who take in charge the treatment of the patient, nurses who ensure the daily follow-up of the patient, the medical secretaries who are interested in the administrative part of the patient sojourn, and also other actors who are transversal to services like physiotherapists, social welfare workers, nutritionists... The PR can then be accessed by different persons with respect of confidentiality restrictions due to the sensitive character of information. The PR is unique and non-modifiable. Indeed the PR recounts the veracious medical story of the patient: diseases, accidents, even bad diagnoses or errors cannot be removed. For all these reasons, the Patient Record is one of the main repository for the knowledge needed in medical activities.

3.2 The Patient Record as a Document

In early MIS, the Patient Record was limited to its "data" part. The medical discipline provides various classifications that were (and are always) used

for formatting the PR in coded data-base suited fields. Among existing classifications, the International Classification of Diseases is the most complete and the most widely used. Medical acts are also codified ; for example, the French CDAM (Catalogue Des Actes Medicaux) allows the coding of acts of different types : radiology, biology, diagnosis acts... At last, medicine databanks are also greatly used. For example, the BCB (Banque Claude Bernard) takes into account all the French medicines and gives coded information about composition, indications, contra-indications, interactions of molecules, and also economic information. Still, the standardization of all the medical information needed for storing the PR in a DB remains unreachable, and in any case, would be a nonsense. Thus, many studies [15,9] have reached the conclusion that a document view of the medical record is less constraining for the medical actors and thus could provide a more viable solution. The next step of this thought is in evaluating the desirable granularity while structurating the electronic patient record document.

3.3 A Semi-structured Document Inside the Patient Record

We can see the PR as an "hyperdocument" composed of various multimedia documents. We categorize these documents in 3 types. In the first type, we can find highly structured documents which contain only structured data (formatted information) and can thus be stored in a database. It is the case for example for some analysis results forms. In the second type, we put structured documents which have a predefined structure for information, but whose information is not formatted [9]. They can be stored using a predefined Document Type Definition (DTD) and belong to the classical SGML-based documents. For example, "end of hospitalization" reports are documents that are normalized for administration needs and do have a pre-defined structure. In the last type, rely semi-structured, or un-structured documents. These documents contain information that cannot be structured in advance, and for which DTDs can hardly be defined. For example, clinicians' notes on patients have a structure which is very specific to each note and cannot be foreseen as a whole. Even so, some experiments have been made for the structuration of this last type of documents [2], [8]. But the result is a stupendous number of DTD so as to cover the domain. Moreover, the aim of these research groups concerns the exchange and communication of information but does not address the indexing and retrieval process of information.

4 Outlines for Information Retrieval in a PR Collection

A PR collection, or pool, can be consulted for finding one specific PR ("the record of patient X") and/or for finding a set of PR corresponding to a special medical case. We here outline how the very specific semantics of the medical information contained in the document can be used, or influence, the building of a query.

4.1 The Chronological Aspect

The information contained in the PR is strongly related to temporal aspects. As addressed before, information is always added in the document, but never modified nor deleted. Indeed, for legal reasons, even erroneous information (diagnosis errors, cancelled prescriptions...) must remain in the PR. The order in which information pieces succeed one another in the document reflects a chronology of facts and thus the medical history of the patient. Moreover, the composition of the PR can be seen as the image of the way the medical staff deals with the patient case, and then becomes a real trace of a reasoning process. For example, the request searching for cases of prescription of a certain drug followed by a particular analysis is in the same time a query on the chronology of acts and on the follow-up of a medical case. But this query is also equivalent to a query on the way the PRs are composed. This case-based searching is frequently used as a lot of studies are carried out that focus on the medical art for epidemiological or for learning purposes. These studies are interested in getting the trace of practitioners' decision making. We think that the PR seen as a ordered list of events can be a good basis for these studies.

4.2 The Contextual Aspect

Another aspect of the medical information is its "context-sensitivity". It is obvious that the mentioning of a drug in a paragraph relating a prescription does not have the same sense as the mentioning of the same drug in a paragraph relating the allergies of the patient. Thus, the meaning, or the semantics, of an information always depends on its writing context. As a consequence, asking for a specific information cannot be understood without making its context clear. This aspect is obviously not specific to Medical Documents. But in this special field, an Information Retrieval system that would be based on keywords would not only be completely useless, but nearly absurd.
To conclude, we can say that the retrieval in our documentary base focuses on the structure of the PRs when querying on patient cases, and uses the structure to be able to understand this query. Despite, the PR remains a semi-structured document. For all these reasons, we need at the same time :

- a non-constraining production process,
- and a retrieval process that relies on the content and the organization of PRs and is robust to the flexibility of the production process.

5 Population of the Documents Collection

In most of the documentary systems, the meta-data associated to a document is built from its author, version number, date of creation and/or a few keywords... But in our context, the PR is written by several authors, it represents the entire medical history of a person, and for this reason it can only exist in a single version, although mentioning different dates. In fact, the element that federates

the information pieces in the PR is the subject of this document, i.e. the patient himself. Each PR in our documents collection thus will be associated to the meta-data "Patient" as it is registered in the AIS. The PR then gathers information of many types: text, images (X-Ray...), graphics (ECG...), animations (echography...), sound (dictated reports...). Any information in a PR may be of any type, independently of its meaning. Analysis results, prescriptions, surgery reports can alternatively be stored as text, sound, video. Moreover, PRs represent a whole which cannot be divided into sub-records according to a criterion like the type of information or its meaning. The PR is filled successively without any other regard than the chronological arrival order of the events generating the information. So, if the PR has no a priori structure, and the information no a priori types, the information still has a well-determined semantics according to the moment or the place they appear. Our proposal is to describe the semantics of information using the concepts of Logical Units and what we call "qualifiers". These two concepts will be used to give a sense to information by giving it a context. In the content, we then distinguish what is usually known as data and what remains "free" information. Logical Units, qualifiers and data are SGML-like tagged in the document.

5.1 The Content

We distinguish "data", which is somehow already part of the AIS or MIS i.e. is already registered in a known DB, against "free information". Data may be selected in any database of the information system : patient or staff databases, medical terms classification, drugs databanks... Depending on their source they may have a strong type, a definition domain, associated methods (in Object-oriented databases). For example, the data "aspirin" is of type "drug", has its definition domain in the "BCB" drug dictionary, and a method called "contra-indications()". At the production time, the author selects a data in the appropriate database. This data is automatically surrounded by its identification, i.e. the reference of the database and the identification of the data itself in its database. This identification can be seen as a reference and is used in this way at the consultation time. `<DATA DBID=BCB OID=124 > aspirin </DATA>` refers to the object "aspirin" identified number 124 from the BCB drugs bank.
Information which is not a data, is "free information". It is captured freely, as in a classic document authoring tool. The information freely captured can be of any type (text, image, graphic, sound or animation) but at present we have restricted our study to textual information, especially for the indexing process.

5.2 The Context

We define a context using two different concepts : "Logical Units" and "qualified text". In medicine, it is classic to distinguish 4 semantic types of information. They are defined by the SOAP acronym [14]:

- S for *Subjective* refers to the information the patient provides ;
- O for *Objective* refers to information the physician discovers (clinical examination or other analysis results) ;
- A for *Assessment* represents the diagnosis of the doctor and more widely his thinking about the case of the patient ;
- P for *Plan* represents all the prescriptions given to the patient ; their aim may be prospective (to verify a hypothesis) or may treat the patient.

They form the 4 Logical Units (LU) of PRs documents. The insertion of any information in a document must be made in the context of one of these logical units. Thus, a LU is created by a person of the medical staff each time he wants to register something about the patient case. He chooses the type of LU he wants to add according to the semantics of its content. The meta-data associated to a LU is its author and its date of creation and are automatically inserted. For example, the author could write the following sentences:

```
<O Author="Dr No" Date="12/12/92"> The patient presents an
undeniable tobacco addiction</O>
<A Author="Dr No" Date="12/12/92"> We may suspect a lung
cancer</A>.
```

Qualifiers make the core originality of our approach. We call "qualified text" a text which can be given sort of a "title", and this without pointing out a logical structure for this text. In fact, what we try to simulate here is the writing form that consists in writing "for example" followed by the examples: the locution "for example" inserted before the information does not give a structure to the text but is used to give a sense, i.e. to qualify, the information that follows. In the same manner, an author could write the following paragraph:

```
<O author="Dr No" Date="12/12/92"> the patient presents an
undeniable <habit> tobacco addiction</habit></O>
<A author="Dr No" Date="12/12/92"> We may <Hypothesis> suspect
a lung cancer</Hypothesis> </A>
```

The two pieces of information "tobacco addiction" and "lung cancer" are respectively said to be a habit and a hypothesis. We can see in this example that the author is free to put e.g. his first qualifier (`<habit>`) in any place of the sentence: after the word "undeniable" or before the word "presents"...: if he wants his text to be correctly qualified, the only requirement is that the words "tobacco" and "addiction" rely in-between the qualifiers. Thus, as a difference with classic SGML tags, qualifiers can be inserted anywhere in the text. The aim is more to provide a way to annotate information in a document instance than to build a model of document. There is no constraint on the type of the information that follows (it could be free information or data of any type), nor on the way qualifier can eventually be nested one into the other (e.g. `<f><g> information </g></f>`). In this last case, we consider that the information is multi-qualified i.e. is qualified by all the qualifiers that surround it. Of course, the qualifiers should not contradict... As mentioned above, qualifiers do not provide a structure for the PRs. This means that there is no SGML-like DTD associated to the PRs. Despite, qualifiers have strong semantics: as stated before,

they must not contradict, their choice may differ from an author to another. This leads to the conclusion that we do not need a simple list of qualifiers to be provided to the authors, but rather a real dictionary where the semantic relationships between qualifiers would be explicit. Thus, we propose to build a thesaurus of pre-defined qualifiers, linked with the traditional synonymy relationships and generalization/specialization relationships (We have thought of bringing a rewriting mechanism in the thesaurus so that a new qualifier could replace the association of two qualifiers, but we have noticed that most of these cases can be solved with the specialization mechanism). In the production process, the thesaurus will be used to help the author in choosing his qualifiers ; then the thesaurus will have a major role in the retrieval process.

As a summarized discussion, we can say that the main differences between SGML tags and qualifiers are:

- SGML-like tags are defined so that their places in the document are regulated. They are provided for structuring purposes. On the contrary, qualifiers are defined for annotation purposes. They do not aim at providing a structure of documents but are intended to explicit the sense of information pieces.
- SGML-like tags are defined in a DTD where composition in the only relationship provided. Qualifiers are stored in a thesaurus which allows for the use of synonymy and generalization/specialization relationships between qualifiers. Composition is not on purpose as any composition is allowed.
- SGML-like tags are defined a-priori and cannot be adapted to each author. Qualifiers may be created and inserted in the thesaurus, so that each author can have an adapted tool.

5.3 Objective versus Subjective Components

At the end-user point of view, we can classify our 4 components (logical units, qualifiers, data and free information) according to their objective or subjective aspect. Objective elements are logical units and data. Objective elements have the following characteristics:

- they are easily identifiable: no misinterpretation can be done about their meaning ;
- they are uniquely identified: a logical unit is either S-typed or O-typed etc.... but can not be overridden;

Subjective elements are qualifiers and free text. They are different from objective elements in the following characteristics:

- their choice depends on the sensitivity of the author: for the same piece of information, two different authors may not use the same qualifiers nor the same words.
- they can be freely defined: new qualifiers can always be inserted in the thesaurus.

5.4 Documents Descriptors

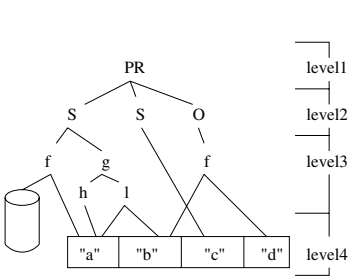
Now that we have described how and with what PRs are composed, we shall try to build up what is called a document descriptor in documentary systems. The descriptor aims to be a summarized description of the documents i.e. what will be indexed to ease the retrieval process.

The tree representation The PR can be represented as a tree in which the nodes are the different components of the document and edges are composition or reference links. This representation is drawn on classic hypertext representations (see Figure 1). This tree can have any depth, but we can distinguish 4 levels that are the following:

- Level 1: the root, i.e. the PR itself
- Level 2: Logical Units: there can be only one node of this type per branch (i.e. LU cannot nest)
- Level 3: Qualifiers: they can nest, so that the depth of this level is any. Qualifiers always contain information: they cannot be placed at a leaf of a tree. In those levels, the nodes are labeled using their identifier in the thesaurus and not their formal names (for example the `<allergy>` qualifier will be referenced in the tree as "f").
- Level 4: Information: See next paragraph for more details.

In these trees, links directed to a data are reference links, while all other links are composition links.

The fourth level Level 4 merges information that originates from the content into 2 parts. One is a list that contains the key-words (items) provided by a full-text indexing mechanism computed on free information. The empty words are eliminated during this parsing process. The second part consists in all the DB used to reference data.




```
<S Author="DrGo", Date = 12/01/97">
  <allergy>...
  <data BDID=BCB OID=24>aspirin</data>
  ...a..</allergy>
  <antecedent>...
   <treatment>...a...</treatment>
   <analysis>..a..b..</analysis>
  </antecedent>...
</S>
<S Author="DrNo" Date="12/12/98">..
  ..c..</S>
<O Author="DrNo" Date="12/12/98">..
  ..<allergy>..d..</allergy> </O>
```

Fig. 1. A document and its tree representation

6 The Information Retrieval Process

There are 2 possible types of information retrieval in a documents collection. The first one is an intra-PR searching. For example the question "what are the prescriptions ordered for patient X ?" concerns one unique document and searches for specific information values in this document. This type of retrieval is useful in the medical following of one patient. The retrieval of the PR is initiated by giving the name (or social number) of the patient. A parsing process inside the PR will probably be launched.
The second type of search, which interests us in this article, is an inter-PR searching (also called cross searching). The questions correspond to the search of a set of cases answering some criteria. The kind of query we treat looks like for example: "select PRs that contains a prescription of drug Y". The answer to such questions is a set of documents. This kind of query is useful for epidemiological studies, case-based teaching... Our approach for information retrieval can be summarized as follows: both documents and queries are represented using a tree-based representation ; the correspondence function between the query tree and documents in the collection is based on pattern matching on flattened trees. In the following we first describe how queries are formulated and then we describe the pattern generation for documents and queries and lastly the matching process.

6.1 Queries by Example

As queries are case-based, they have the form of documents and thus it is straightforward to propose a query-by-example language. This means that the requester formulates his query in the same manner as an author fills a document. The query language uses the concepts we previously described for documents: LU, qualifiers, data and free information. If the user is searching for "cases where a patient announced an allergy to A treated with B and where more investigations showed an allergy to C", then the query stated is :

```
<Q>
<S> <allergy>A</allergy> <treatment> B</treatment> </S>
<O> <allergy>C</allergy></O>
</Q>
```

Indeed:

- Information that has been collected during an interview with the patient is necessarily in a S-type LU: that is why the selected PRs will have their allergy and treatment information in a S-type LU.
- Information computed during a medical act like clinical examination or analysis is "objective" and then relies in a O-type LU. The query does not mention a special type of act to be selected, that is why there is no more qualifier in the query than the `<allergy>` one.
- The order of LU is meaningful because the analysis part comes to confirm what the patient said.

These three points are detailed here to explain how the requester builds his request ; but it is obvious that he builds this request naturally, without having to explicitly think of all the steps he makes. Thus the queries are formulated as documents. They can be represented using the tree-representation previously described. The tree for the above example is given in Figure 2 :

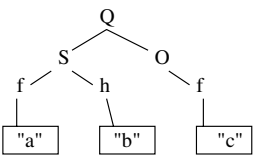


Fig. 2. Tree representation of a query

6.2 Trees Implementation

Documents and queries can be described as trees, as shown in previous paragraph. To implement the index on documents and to make pattern matching between a query and the documents, we use a flattened version of trees.

Flattening principles The aim is to produce a string of tokens corresponding to a flat representation of each tree. The flattening process simply consists in a deep-first parsing of a tree. It follows 3 rules :

- encountering a start tag for a LU or a qualifier node results in the creation of a starting token as `'s' <node identifier>` (e.g. sf) ;
- encountering an item or data results in the creation of a token for this leaf, which is `<leaf identifier>`;
- encountering an end tag for a LU or a qualifier node results in the creation of a "ending" token as `'e' <node identifier>`.

Thus, the token string produced is a synthesized version of the document where identifiers of LU, qualifiers, items and data only remain (empty words do not appear).

The documents index Each document descriptor is linearized as described above. Our system also contains an index of the items and data used; each index entry addresses a list of pointers to the flattened descriptors in which the entry appears. In this way, each document descriptor is pointed to by as many index entries as the number of items and data it contains. Similarly, each descriptor addresses its corresponding PR in the documents collection. This index speeds up the query answering as it restricts the set of documents to submit to the pattern matching process to the set of documents containing the items and data contained in the query.

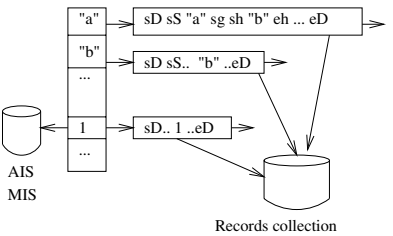


Fig. 3. The documents collection indexing

The query pattern The answers to a query must contain the same LU within the items, identically qualified, and in the same order. Nevertheless, relevant documents may also contain other components, and this in-between the terms of the query. In figure 4, the document drawn in (c) is an answer to the (a) query. In other terms, the tree of the query must be a sub-tree of the document

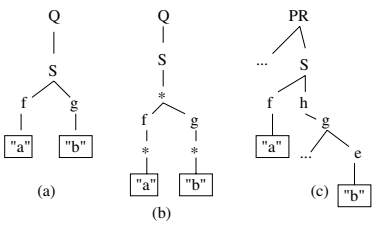


Fig. 4. Query patterns

tree. Thus, we have to add "virtual" nodes in the query tree to represent the fact that the qualifiers and items we are looking for can stand in variable depth and/or width in the matching trees. This step is shown on Figure 4 (b). The pattern to be found in relevant documents has to be modified to ensure it takes care of the depth and width variability of the answers. It is re-written into the following pattern: sD * sS * sf * a * ef * eS * eD where "*" is any series of tokens, i.e. any sub-tree of the document. Let us notice that the existence and depth or width of these sub-trees do not allow us to conclude on the relevance degree of the documents. As a matter of fact, the depth and width of the trees depends on the precision how the documents and queries have been produced, characteristic which is totally dependent on the author's way of working. A meticulous author includes much more context information and content than a more lazy writer.

6.3 Evaluation of Queries

We shall now explain the main principles of our query process. Remember that at this step, we have to match a query pattern containing virtual tokens to a descriptors base indexed by a list of items and/or data.

Pattern matching The first step is to search the entries in the index that correspond to the items or data tokens of the query. It restricts the number of pattern matching processes to attempt. Then we have to compare each descriptor pointed out by the entry with the pattern of the query. The algorithm we use is merely close to the classic Unix "grep" matching a regular expression through the lines of a text file.

Query reformulation The first step allows us to compute the more relevant documents i.e. the ones that contain exactly the same tokens, in the same order, but with various depth or width. In the second step, we have to find "similar" documents i.e. documents that were not qualified exactly like the query asked for, but with qualifiers that are semantically close to them. In that purpose, we have to replace the qualifiers of the query by the qualifiers mentioned in the thesaurus as synonyms, or linked with generalization/specialization links. The use of specialized qualifiers does not modify the accuracy of the answer, whereas the use of generalized terms reduces it. At this stage of our study, we did not focus on the calculation of accuracy. It is obvious that this problem will have to be addressed deeper because synonymy, generalization and specialization are a direct consequence of the way documents are produced and qualified.

Pre-selection of data As stated earlier, queries are formulated like documents. This means that items and/or data appear in the query document. If items are easily typed, data may need to be selected in databases. For example a query like "looking for PRs containing a plan mentioning a drug from the antidepressive class" will first need a classic selection query in the drugs database. The answer of this first selection is a set of values. Then one document query has to be created for each element of this answer.

7 Conclusion

In this article, we have proposed a new way for the authoring and selection of documents. The logical structure of documents is reduced to the use of 4 different logical unit types, while information pieces can be qualified by authors. Qualification is a very flexible process based on the selection of qualifiers in a thesaurus which also contains synonymy and generalization/specialization relationships between qualifiers. Querying documents consists in query-by-example. The query focuses on the chronology and the content of the events related by the PR, thus consisting in a tree pattern matching. For the pattern matching process,

documents and queries are represented with documents descriptors which can be compared. A documents index allows for speeding up the matching process as it reduces the documents descriptors collection to the subset of documents containing the items and data present in the query. The next step of this study is to provide an accuracy measure of answered documents. We first have to define an order in the query reformulation tools and secondly we would like to study in which measure providing documents that answer only one part of the query can be relevant. The definition of the end-user tool and interface of such a system can also be a good research deal.

References

1. S. Abiteboul, D. Quass, J. McHugh, J. Widom, JL Wiener: The Lorel query language for semi-structured data, in Journal of Digital Libraries, vol. 1,num. 1, 1997 62
2. L. Alschuler, al.: The Kona proposal, http://www.mcis.duke.edu/standards/HL7/ 64
3. M. Bryan : SGML, an Author's Guide to the Standard Generalized Markup Language, Ed. Addison-Wesley, 1988 61, 62
4. G. Salton, M. Mc Gill: SGML, Automatic text processing, Ed. Addison-Wesley, 1989 62
5. P. Buneman, S. Davidson, G. Hillebr : A query language, optimization techniques for unstructured data, in proc. of ACM SIGMOD Conference, june 1996 62
6. V. Christophides , S.Abiteboul, S. Cluet, M. Scholl : From structured documents to Novel query facilities, in proc. of ACM SIGMOD Conference, may 1994, Minneapolis 62
7. A. Deutsch, M. Fernandez, D. Florescu, A. Levy, D. Suciu : A Query Language for XML, in proc. International WWW Conference 1999 62
8. HL7 SGML/XML SIG :SGML/XML as an interchange format for HL7 V2.3 messages, http://www.mcis.duke.edu/stards/HL7/commitees/sgml/WhitePapers/HL7V2 64
9. F. Laforest, S. Frenot, A. Flory: A hypermedia-based medical records management system, in proc. International Conference MedInfo, August 1998, Seoul (Korea) 64
10. J. Le Maitre, E. Murisasco, M. Rolbert: SgmlQL, a language for querying SGML documents, in proc. 4th European Conference on Information Systems, 1996, Lisbonn, Portugal 62
11. C.J. Mac Donald, W.M.Tiernet: Computer-stored medical records: future role in medical practice, in Journal of American Medical Association,1988 63
12. M. Stonebraker, H. Stettner, N. Lynn, J. Kalash, A. Guttman: Document processing in a relational database system, in journal ACM TOIS, vol. 1, num. 25, April 83 62
13. W3C : eXtensible Markup Language (XML), XML-Link, XS (XML-DSSSL-o), 1997 61, 62
14. L.L. Weed : Medical records, medical education, patient care : the problem oriented record as a basic tool, The Press of Case Western Reserve University 1969, Cleveland, Ohio 66

15. P. Zweigenbaum , al. : From text to knowledge: a Unifying Document-Centered View of Analyzed Medical Language, in journal Methods of Information in Medicine, 1998 64
16. D. Konopnicki O. Shmueli : Information Gathering in the World Wide Web: the W3QL Language, the W3QS System, in ACM Transactions on Database Systems, vol. 23, num. 4, December 98

Word-Based Compression Methods and Indexing for Text Retrieval Systems

Jiří Dvorský[1], Jaroslav Pokorný[2], and Václav Snášel[1]

[1] Computer Science Department, Palacky University of Olomouc, Tomkova 40, 779 00 Olomouc, Czech Republic
{jiri.dvorsky,vaclav.snasel}@upol.cz
[2] Department of Software Engineering, Charles University, Malostranske namesti 25, 118 00 Prague, Czech Republic
pokorny@ksi.ms.mff.cuni.cz

Abstract. In this article we present a new compression method, called WLZW, which is a word-based modification of classic LZW. The modification is similar to the approach used in the HuffWord compression algorithm. The algorithm is two-phase, the compression ratio achieved is fairly good, on average 22%-20% (see [2],[3]).
Moreover, the table of words, which is side product of compression, can be used to create full-text index, especially for dynamic text databases. Overhead of the index is good.

1 Word-based compression

Data compression is widely used in text (or full-text) databases to save storage space and network bandwidth. In full-text retrieval three resources are at disposal: secondary storage, retrieval time and primary memory. Both indexing and compression methods are trade-off between the use of these resources. In typical full-text database, various auxiliary structures are adepts for compression in addition to the main text. They include at least a text index, a lexicon, and disk mappings [1].

Many compression methods can be used to reduce size of all these data. Relatively more advanced methods are based on coding words as basic unit. Good compression can be achieved by coding words on their frequency. We could cite word-based adaptive Huffman coding introduced in [6] and HuffWord algorithm given in [12]. Another experiments have been done with word-based LZW by Horspool and Cormack [6],[10]. Our algorithm differs to the word-based LZW presented there in the following:

1. WLZW uses only one data structure for the lexicon (table) of strings comparing to two tables Atable (string whose initial substring is word) and Ptable (strings whose initial substring is non-word),
2. It is two-phase because this property is not restricting in full-text databases,
3. a single data structure for the lexicon is usable as text index,

Eder et al. (Eds.): ADBIS'99, LNCS 1691, pp. 75-84, 1999.
© Springer-Verlag Heidelberg Berlin 1999

4. the length of words and non-words is restricted (this feature improves the compress ratio achieved),
5. empty words or non-words are inserted into text, and
6. selected non-word is eliminated from input.

In section 1.1, we introduce shortly the LZW algorithm, and its implementation problems. Section 2 contains our extending LZW to WLZW, i.e. the compression and decompression algorithms are used presented in detail and discussed from implementation point view. Some aspects of inverted files for dynamic collections are given in section 3. Experimental results are summarized in section 4. Finally, we conclude with some practical issues concerning usage of WLZW in text retrieval.

1.1 LZW — a short summary

Most of techniques based on adaptive dictionaries have their roots in two algorithms developed in 1977 and 1978, respectively. They are known as LZ77 [13] and LZ78 [14] (see also e.g. [9],[8]). There is a number of ways the LZ78 algorithm can be modified. The most well-known modification is by Terry Welch. His refinements to the algorithm were published in 1984 and are known as LZW [11].

The algorithm is simple. In a nutshell, LZW compression replaces strings of characters with single codes. This means that the lexicon cannot be empty at the start and all one-characters strings are already in the lexicon, i.e. a table that associates a unique integer with each string. LZW does not do any analysis of the incoming text. Instead, it just adds every new string of characters it sees to a table of strings. Compression occurs when a single code is output instead of string of characters. To encode the next segment of source text, the compression algorithm reads the longest possible sequence of characters that comprises a string in the table. If the string that was read was w and C is the following character in the source text, the new string wC is added to the lexicon and the next unused number is associated with the string.

The code that the LZW algorithm outputs can be any of arbitrary length, but it must have more bits in it than single character. The first 256 codes (when using eight bit characters) are by default assigned to the standard character set. The remaining codes are assigned to strings as the algorithm proceeds. The sample program runs as shown with 12 bit codes. This means codes 0–255 refer to individual characters, while codes 256–4095 refer to substrings.

The decompression algorithm does not need to pass the string table. The table can be build exactly as it was during compression, using the input stream as data. This is possible because the compression algorithm always outputs wC components of code before it uses it in the output stream. This means that compressed data is not burdened with carrying a large string translation table.

2 WLZW algorithm

The creation of the WLZW algorithm was inspired by the work of Alistair Moffat, who published their HuffWord algorithm, see ref. [12]. The WLZW algorithm

is based on the idea that sequences of words repeat in text. The repetition arises from the structure of a natural language. This is similar to LZW where compression is based on the assumption that repetitions of sequences of characters occur in text (see [4],[5],[7]).

The alphabet of the WLZW compression algorithm consists of words and non-words. The adaptation of LZW to word compression raises a problem with the initialization of the compression dictionary. If the classic approach is used then at the beginning of compression the dictionary is filled by characters with codes from 0 to 255, since an alphabet with 256 characters is used. For word compression, the size of the alphabet is not known beforehand. Horspool and Cormack implemented the word-based LZW algorithm using only a single pass through the text. This method requires an escape mechanism for signalling and storing characters which have not yet been used.

The algorithm we have implemented uses two passes because of the archival nature of textual databases. The text of a document is stored only once and is then read multiple times. This makes it possible to choose a better compression ratio, even at the price of two passes and lengthier compression. Another consideration is that computers used for constructing collections of documents are usually more powerful than client computers accessing the finished collections.

2.1 Definition of words and non-words

The alphabet of the WLZW consists of words and non-words. We called them *tokens* together. A word is defined as maximal string of alphanumeric characters (letters and digits) and non-word is defined as maximal string of other characters (punctuations and spaces). For example sentence

`In␣the␣beginning␣God␣created␣the␣heaven␣and␣the␣earth.`

can be divided into word, non-word sequence: "`In`", "␣", "`the`", "␣", "`beginning`", "␣", "`God`", "␣", "`created`", "␣", "`the`", "␣", "`heaven`", "␣", "`and`", "␣", "`the`", "␣", "`earth`", "`.`" (where ␣ represents space).

Length of the words for natural languages is bound by several tens of characters (for Czech language about 30), but length of some non-words may be several hundreds of characters — some plain ASCII formatted texts contain long string of spaces, or semigraphics in such texts. Storing such strings in fulltext index's files is much more complicated then storing fixed-length strings. In opposite, to achieve the best compression ratio the lengths of tokens must be the longest possible. Experimental results shows that compression result depends on length of word and no so much on length of non-word. Thus the length of tokens can be restricted to the length of the longest word, which is considered in fulltext queries. Long sequences of spaces are divided into several smaller ones. Cardinality of the alphabet is reduced too, because there is only limited number of tokens.

It is clear that words and non-words from input strictly alternate. The alternating of tokens is important piece of information. With this knowledge kind of

next token can be predicted. But if the lengths of tokens are restricted alternating of tokens is corrupted, because some tokens were divided in several parts of same type. To save alternating of tokens two special tokens are created. They are *empty word* and *empty non-word*. They contain no character. Empty word is inserted between two non-words and empty non-word between two words. In this manner alternating of tokens is saved for all sequences of tokens.

2.2 Compression

Compression takes two phases:

1. Lexical analysis - the input text is divided into words and non-words. This is how the LZW compression alphabet is created.
2. The compression itself.

When using two passes variant it is necessary to store the table of words and non-words together with the compressed text. However, this table can be transformed to make it suitable for query evaluation. To implement the WLZW algorithm, the following modifications to the LZW algorithm are needed:

- In character-based compression, ordinal numbers of characters are used directly as the entries in the compression dictionary. Storing strings of characters forming words and non-words directly in the dictionary would be impractical and memory-consuming. That is why words and non-words are stored in a special table where they are numbered in ascending order. In this way a unique identification number is assigned to each of them. Only their numbers are listed in the compression dictionary. During compression, the token read from input is looked up in the table first and subsequent operations only use its identification number.
- 4096 phrases, the standard range of the dictionary, are obviously not sufficient, because the alphabet (i.e. the number of distinct words and non-words) is usually higher - typically from several thousands up to several tens of thousands. We chose 2^{20} phrases as the upper limit; 4096 phrases were left as the lower limit. Since we use variable code length, the length of a coded phrase fluctuates between 12 and 20 bits. We decided that it was not necessary to use a dictionary with a million phrases. In character-based LZW the ratio of cardinality of the input database to the size of the dictionary is 1 : 16 (256 : 4096). We tried to maintain this ratio. So our test implementation of the algorithm automatically sets the size of the dictionary (unless explicitly stated otherwise) to 16 times the closest power of 2 greater than the cardinality of the alphabet used (with respect to the upper limit of the size of the compression dictionary, fixed to 2^{20}). It shows that this estimation provides the capability of setting the optimum size of the compression dictionary.

We can see in sample sentence from previous section, that the words are separated by non-words, in our case by nine single spaces and only one comma. Horspool and Cormack consider these single spaces as a special part of words

and don't code them. Spaces are written to decompressed file only if no other non-word follows after decompressed word. This principle reduces number of tokens and produces smaller compressed files. Eliminating of single space can be generalized to eliminating some other non-word. We called them *victim*.

In compression phase input token stream is converted to strictly alternating stream of words and non-words. Victim is deleted from stream in the next step. Finally, this stream is compressed.

In decompression phase codes of phrases are expanded in sequence of tokens. Deleted victim is recognized as error in alternating of words and non-words in sequence. When victim is inserted, tokens, except empty tokens, are saved to output file.

How to find appropriate victim? Victim should appear frequently in text, to achieve high probability, that non-word between two words is just victim. In this way big number of tokens is eliminated. But let us consider length of victim too. If length of victim grows more characters of input text aren't represented as tokens in compression algorithm itself.

We suggest to use following weight function:

$$w_x = \sum_{i=1}^{3} \sum_{j=1}^{3} c_{ij} f_x^{i-1} l_x^{j-1}$$

where

- w_x is weight of non-word x
- $c_{ij} \in R$, $i, j \in \{1, 2, 3\}$. Initially we use matrix

$$C = \begin{pmatrix} 0\,0\,0 \\ 1\,0\,0 \\ 0\,0\,0 \end{pmatrix}$$

- f_x is the frequency of non-word x
- l_x is the length of non-word x

Victim is choose as non-word with maximal weight.

2.3 Decompression

Decompression is similar to classic LZW. It takes three phases:

1. loading table of tokens
2. initialization of the decompression dictionary
3. decompression loop

Just as in compression, identification numbers of words and non-words are used in the decompression dictionary instead of strings of characters. Phrases are made up of these numbers. Only during the dump of completely decompressed phrases are strings corresponding to words and non-words stored in the output file, after being translated step by step using the table.

In classic LZW decompression dictionary is initialized by the input alphabet. This part of the dictionary is static and no memory has to be allocated for it. For character-based compression this space is negligible, but for word-based compression it offers the possibility of significant space saving. That is why the initial part of the dictionary is simulated by a program code in our implementation.

3 Inverted file indexing

An index is mechanism for locating given term in a text. There are many ways to achieve this. In environment with large text, there are three most suitable structures are *inverted files*; in normal English usage as *concordances*; *signatures* and *bitmaps*. Inverted file and bitmap structure both require a *lexicon* or vocabulary – a list of all terms that appear in the database.

Let us now define exactly what we mean by an inverted file index. An *inverted file* contains, for each term in the lexicon, an *inverted file entry* that stores a list of pointers to all occurrences of that term in the main text, where each pointer is, in effect, the number of document in which that term appears.

Our inverted file entry contains tn – term number obtained from token table (lexicon), four bytes long, d — document number, four bytes long and $f_{d,t}$ term-document frequency, two bytes long. Inverted file entry have 10 bytes. Our inverted file entry has same properties as in [12].

A query involving a single term is answered by scanning its inverted file entry and retrieving every document that it cites. For conjunctive Boolean queries of the form "term AND term ... AND term", the intersection of terms's inverted file entries is formed. For disjunction, where the operator is OR, the union is taken; and for negation using NOT, the complement is taken. The inverted file entries are usually stored in order of increasing document number, so that these various merging operation can be performed in a time that is linear in the size of the lists.

The *granulity* of an index is the accuracy to which it identifies the location of a term. A coarse-grained index might identify only a block of text, where each block stores several documents; an index of moderate grain will store locations in terms of document number; while fine-grained one will return a sentence or word number or perhaps even a byte number.

Throughout the following discussion, it will be assumed that the index is a simple document-level one. In fact, given that a "document" can be defined to be very small unit such a sentence or verse (as it is for the *Bible* database), in some ways word-level indexing is just an extreme case in which word is defined as a document.

3.1 Dynamic collections

Throughout the previous description, we have assumed that the database is static. However, it is rare for a database to be truly static. The problem of

dynamic collections cannot be ignored. A collection can be dynamic in one of two ways. First, it might provide an "insert" operation that appends a new document to an existing collection but not does not change any of the current documents. More radically, it might also be necessary to support an "edit" operation that allows documents to be altered and perhaps even removed.

Expanding the text Consider operation of inserting a new document. First, the text of the collection must be expanded. Our experimental system support an "append" operation on previously created files, so it is relatively straightforward to append the new document to the text of the collection. Care must be taken, when document are being compressed, and the compression algorithm must be able to cope with unseen token. The new token must be inserted to the table of tokens.

Expanding the index Consider an inverted file. Since each newly-inserted document contains many terms, the inverted file must support *multi-point* expansion rather than a simple append mode of expansion. A simple file is not appropriate tool for that purpose. Data structure that allows inserting and deleting records in time proportional to number of records is B-tree.

3.2 Index construction

Short description of algorithm which produce an inverted file for a collection of documents.

1. /* Initialization */
Create an empty table of tokens T.
Create an empty inverted file B.
2. /* Phase one — collection of term appearances */
For each document D_d in the collection, $1 \leq d \leq N$
(a) Read D_d, parsing it into index terms
(b) For each index term $t \in D_d$,
i. Let $f_{d,t}$ be the frequency in D_d of term t.
ii. Search T for t. If t is not in T, insert it.
iii. Let tn be token (term) number of term t. tn is obtained from table T.
iv. Insert a record $\langle tn, d, f_{d,t} \rangle$ to B.
3. /* Phase two — compression of documents */ For each document D_d in the collection, $1 \leq d \leq N$
Compress document d (see section 2).

4 Experimental results

Too allow practical comparison of algorithm, experiments have been performed on some real-life document collections. Two collections have been used in preparing this article.

King's James Bible – Canterbury Compression Corpus (large files)
Czech trade law – collection of documents in Czech language.

Some statistics for these are listed in table 1. In table 1, and throughout the article, N is used to denote the number of documents in some collection, n is the number of distinct terms that appear; F is the total number of terms in the collection; and f indicates the number of pointers, that appear in document level index — the "size" of index.

		Collection	
		Bible	*Trade*
Documents	N	30383	276
Number of terms	F	767855	1840617
Distinct terms	n	12473	74500
Index pointers	f	599979	358394
Total size (Mbyte)		3.89	19.5
Avg. number of terms in doc.		25	6667
Avg. number of pointers in doc.		20	1299

Table 1. Parameters of document collections

Table 2 show index and text sizes for two test collections, both as raw amount in megabytes and as percentage of the original text. For small text, its compressed version, occupied 22.1 percent of original size; for larger text only about 20 percent. The index consumed about 55.5 percent for *Bible* due to very small document. For *Trade* collection index have acceptable size – 24.6 percent (documents are longer than the *Bible* is). Terms which are included in the index are only folded to lower case, no stop-list nor stemming is used, in other words size of our index is maximal and can be reduced by mentioned mechanism.

Fulltext index can be compressed by using Bernoulli model and the Golomb code. This provides fast decoding with acceptable compression (see [12]).

Auxiliary files are both very small. Included in this are the files for token table (lexicon), files storing addresses of documents in compressed text database etc.

Tests were done on Pentium II/233Mhz with 64MB of RAM. Program was compiled by MS Visual C++ 5.0 as 32-bit console application under MS Windows NT 4.0 Workstation.

5 Conclusion

Compression methods suitable for use in textual databases have certain special properties:

– these methods must be very fast in decompression;

	Collection	
	Bible	*Trade*
Compressed text (MBytes)	0.86	3.9
Percentage of input text	22.1	20
Auxiliary files (Mbytes)	0.1	0.3
Percentage of input text	2.6	1.5
Index size (Mbytes)	11.4	5.9
Compressed index (Mbytes)	1.2	0.6
Percentage of input text	30.8	3.1
Total retrieval system (Mbytes)	2.16	4.8
Percentage of input text	55.5	24.6
Input text (Mbytes)	3.89	19.5

Table 2. Index on sample texts

	Collection	
	Bible	*Trade*
Max. number of record in page	100	100
Number of pages	10227	5273
Number of records	599979	358394
Slack space	41%	32%

Table 3. B-tree parameters

– the information necessary for decompression should be usable for text searching.

Experimental implementation of the WLZW method proved that the method proposed by us satisfies these requirements.

It is important to realise that this method doesn't actually depend on text encoding. This means that it performs successfully for text encoded in UNICODE as well. This property is very important when compression of bilingual text is concerned.

The compression ratio achieved is fairly good, on average 25-20% (2% in extreme cases - RTF document). Another experimental results are given in [2],[3].

Our next step is to use this method in a full text system. First results of index mechanism show good performance, especially for large document collections.

References

1. T. C. Bell. et al.: *Data Compression in Full-Text Retrieval Systems,* Journal of the American Society for Information Science. 44(9), 1993, pp.508-531.
2. J. Dvorský, V. Snášel, J. Pokorný: *Word-based Compression Methods for Text Retrieval Systems.* Proc. DATASEM'98, Brno 1998
3. J. Dvorský, V. Snášel, J. Pokorný. *Word-based Compression Methods for Large Text Documents.* Data Compression Conferencs - DCC '99, Snowbird, Utah USA.

4. W. F. Frakes, R. B. Yates Ed.: *Information Retrieval, Data Structures & Algorithms.* Prentice Hall 1992
5. G. H. Gonnet, R. Beaza-Yates: *Handbook of Algorithms and Data Structures.* Addison-Wesley Publishing, 1991
6. R. N. Horspool, G. V. Cormack: *Construction Word-based Text Compression Algorithms,* Proc. 2nd IEEE Data Compression Conference, Snowbird, 1992
7. D. Húsek, M. Krejčí, V. Snášel: *Compress methods for Text Databases.* Cofax 96, Bratislava. (in Czech)
8. B. Melichar, J. Pokorný: *Data Compression.* Survey Rep. DC-94-07. Czech Technical University, Praha 1994
9. K. Sayood: *Introduction to data compression.* Morgan Kaufmann Publishing, 1996
10. D. Salomon: *Data Compression,* Springer Verlag, 1998
11. T. A. Welch: *A Technique for High-Performance Data Compression.* IEEE Computer 17, 6, 1984, pp. 8–19
12. I. H. Witten, A. Moffat, T. C. Bell: *Managing Gigabytes: Compressing and Indexing Documents and Images.* Van Nostrand Reinhold, 1994.
13. J. Ziv, A. Lempel: *An universal algorithm for sequential data compression,* IEEE transac. on Information Theory, Vol. IT-23, No.3., 1977, pp.337-343
14. J. Ziv, A. Lempel: *Compression of individual sequences via variable-rate coding,* IEEE transac. on Information Theory, Vol. IT-24, No.5., 1978, pp.530-536

Processing of Spatiotemporal Queries in Image Databases*

Theodoros Tzouramanis, Michael Vassilakopoulos, and Yannis Manolopoulos

Data Engineering Lab, Department of Informatics
Aristotle University, 54006 Thessaloniki, Greece
theo@delab.csd.auth.gr
manolopo@csd.auth.gr
mvass@computer.org

Abstract. Overlapping Linear Quadtrees is a structure suitable for storing consecutive raster images according to transaction time (a database of evolving images). This structure saves considerable space without sacrificing time performance in accessing every single image. Moreover, it can be used for answering efficiently window queries for a number of consecutive images (spatio-temporal queries). In this paper, we present three such temporal window queries: strict containment, border intersect and cover. Besides, based on a method of producing synthetic pairs of evolving images (random images with specified aggregation) we present empirical results on the I/O performance of these queries.

Keywords: Spatio-temporal databases and queries, transaction time, access methods, indexing, B^+trees, linear region quadtrees, overlapping, time performance.

1 Introduction

Several spatial access methods have been proposed in the literature, for storing multi-dimensional objects (e.g. points, line segments, areas, volumes, and hypervolumes) without considering the notion of time. These methods are classified in one of the following two categories according to the principle guiding the hierarchical decomposition of data regions in each method: data space hierarchy and embedding space hierarchy. The book by Samet [10] and the recent survey by Gaede and Guenther [3] provide excellent information sources for the interested reader.

On the other hand, temporal access methods have been proposed to index data varying over time without considering space at all. The notion of time may be of two types: transaction time (i.e., time when the fact is current in the database and may be retrieved) and valid time (i.e., time when the fact is true in the modeled reality) [4]. A temporal DBMS would support at least one of these

* Research performed under the European Union's TMR Chorochronos project, contract number ERBFMRX-CT96-0056 (DG12-BDCN).

Eder et al. (Eds.): ADBIS'99, LNCS 1691, pp. 85–97, 1999.
© Springer-Verlag Berlin Heidelberg 1999

two types of time. A wide range of access methods has been proposed to support multi-version/temporal data by keeping track of data evolution over time. For excellent recent surveys on temporal access methods see [7,9].

Until recently the field of temporal databases and spatial databases remained two separate worlds. However, modern applications (e.g. geographical information systems, multimedia systems, scientific and statistical databases, such as medical, meteorological, astrophysics oriented databases) involve the efficient manipulation of moving spatial objects, and the relationships among them. Therefore, there is an emerging growing need to study the case of "spatio-temporal databases". According to the first attempt towards a specification and classification scheme for spatio-temporal access methods [13], up until the time it was written, only four spatio-temporal indexing methods had appeared in the literature: 3D R-trees [12], MR-trees and RT-trees [18], and HR-trees [8]. All these methods are extensions of the R-tree, which is based on the "conservative approximation principle", i.e. spatial objects are indexed by considering their minimum bounding rectangle (MBR). These methods are not suitable for representing regional data, in cases where a lot of empty ("dead") space is introduced in the MBRs, since this fact decreases the index ability to prune space and objects during a top-bottom traversal.

In [15], a different paradigm was followed, that of quadtrees. Quadcodes were used to decompose image data in an exact (i.e. non-rough) manner. As a result, a new spatio-temporal structure was presented, named Overlapping Linear Quadtrees, suitable for storing consecutive raster images according to transaction time (a database of evolving images). This structure is based on linear quadtrees which are enhanced by using the overlapping technique in order to avoid storing identical sub-quadrants of successive instances of image data evolving over transaction time. In [15] it is shown by experimentation with synthetic regional data that the new structure saves considerable space, without sacrificing time performance in accessing every single image. Moreover, in [15] an abstract algorithm for answering temporal window queries with Overlapping Linear Quadtrees is presented.

In the present paper, we elaborate on spatio-temporal queries that can be answered efficiently with this structure. More specifically, we present three temporal window query processing algorithms: strict containment, border intersect and cover. We also report on I/O efficiency results of experiments performed with these algorithms. The experiments were based on synthetic pairs of evolving images. The first image of each pair was formed according to the model of random images with specified aggregation of black regions [6]. The second image of each pair was formed by random change of pixels, a rather pessimistic method of image changing. In real situations it is expected that the above algorithms will perform even better.

The rest of the paper is organized as follows. Section 2 describes the building blocks of the implementation of Overlapping Linear Quadtrees. Section 3 provides a detailed description of the three algorithms that use the new structure and answer spatio-temporal queries. Section 4 presents the experimental setting

and reports on the I/O performance of these algorithms in terms of the number of disk assesses. Section 5 provides conclusions and suggestions for future work directions.

2 The Spatiotemporal Structure

The notion of overlapping consecutive instances of access methods has been mentioned in the previous section. Except for the cases of MR-trees and HR-trees, overlapping has been also used in a number of occasions, where successive data snapshots are similar. For example, it has been used as a technique to compress similar text files [1], B-trees and B^+trees [2,5,14], as well as main-memory quadtrees [16,17]. In this section, first we make a short presentation of region quadtrees, and second we describe the application of overlapping to secondary memory quadtree variations.

2.1 Region Quadtrees

The region quadtree is the most popular member in the family of quadtree-based access methods. It is used for the representation of binary images. More precisely, it is a degree-four tree. Each node corresponds to a square array of pixels (the root corresponds to the whole image). If all of them have the same color (black or white), then the node is a leaf of that color. Otherwise, the node is colored gray and has four children. Each of these children corresponds to one of the four square sub-arrays to which the array of that node is partitioned. For more details regarding quadtrees see [10]. Figure 1 shows an 8×8 pixel array and the corresponding quadtree. Note that black (white) squares represent black (white) leaves, whereas circles represent gray nodes.

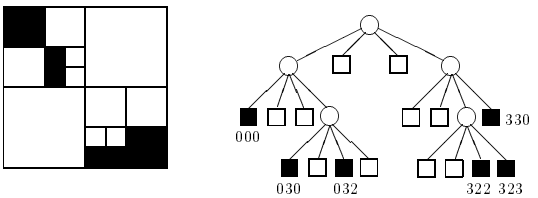


Fig. 1. An image, its Quadtree and the linear codes of black nodes

2.2 Overlapping Linear Quadtrees

Variations of region quadtrees have been developed for secondary memory. Linear region quadtrees are the ones used most extensively. A linear quadtree representation consists of a list of values where there is one value for each black node of the pointer-based quadtree. The value of a node is an address describing the position and size of the corresponding block in the image. These addresses can

be stored in an efficient structure for secondary memory (such as a B-tree or one of its variations). The most popular linear implementations are the FL (Fixed Length), the FD (Fixed length – Depth) and the VL (Variable length) linear implementations [11]. In the FD implementation, the address of a black quadtree node has two fixed size parts: the first part denotes the path (directional code) to this node (starting from the root) and the second part the depth of this node. In Figure 1, one can see the directional code of each black node of the depicted tree.

Each quadtree, in a sequence of quadtrees modeling time evolving images, can be represented in secondary memory by storing the linear FD codes of its leaves in a B^+tree. The structure of Overlapping Linear Quadtrees is formed by overlapping consecutive B^+trees, that is by storing the common subtrees of the two trees only once (for more details regarding this structure, see [15]). Since in the same quadtree two black nodes that are ancestor and descendant cannot co-exist, two FD linear codes that coincide at all the directional digits cannot exist neither. This means that the directional part of the FD codes is sufficient for building B^+trees at all the levels. At the leaf-level, the depth of each black node should also be stored so that images are accurately represented and that overlapping can be correctly applied. The above part of Figure 2 depicts the B^+trees that correspond to two region Quadtrees and the below part depicts the resulting overlapped linear structure. Note that in Overlapping Linear Quadtrees there is no extra cost for accesses in a specific linear quadtree.

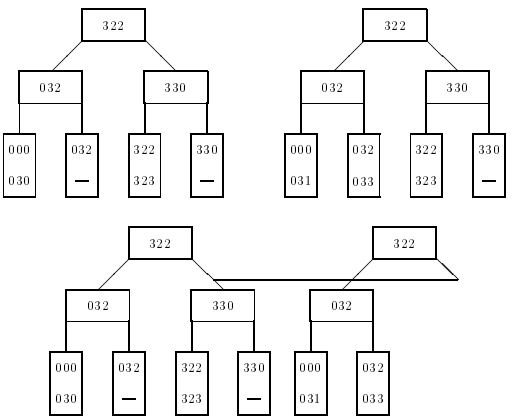


Fig. 2. Two B^+trees storing linear quadtree codes and the corresponding linear overlapped structure.

All nodes of Overlapping Linear Quadtrees have an extra field, called "StartTime", that can be used to detect whether a node is being shared by other trees. We assign a value to StartTime during the creation of a node and there is no need for future modification of this field. Moreover, leaf-nodes have one more extra field, called "EndTime", that is used to register the transaction time when a specific leaf changes and becomes historical.

In order to keep track of the image evolution (in other words, the evolution of quadcodes) and efficiently satisfy spatio-temporal queries over the stored raster images, we embed some additional horizontal pointers in the B^+tree leaves. This way there will be no need to top-down traverse consecutive tree instances to search for a specific quadcode, thus avoiding excess page accesses. More specifically, we embed two forward and two backward pointers in every B^+tree leaf to support spatio-temporal queries. The F-pointer of a node points to the first of a group of leaves that belong in a successive tree and have been created from this node after a split/merge/update. The FC-pointers chain this group of leaves together. The B and BC pointers play analogous roles when traversing the structure backwards.

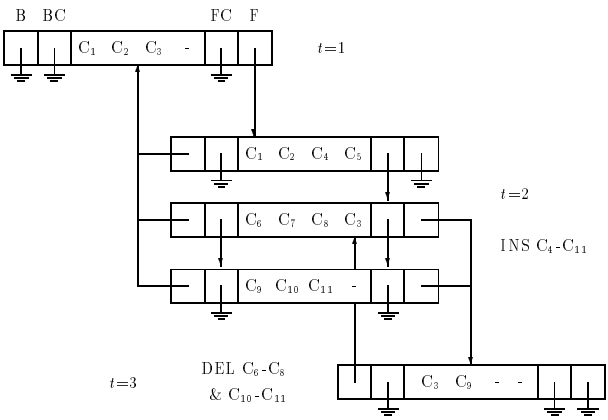


Fig. 3. Forward and backward chaining for the support of temporal queries.

Figure 3 shows how the leaves of three successive B^+trees can be forward- and backward-chained to support temporal queries. The leaf on the left-top corner of the figure corresponds to the first time instant, t=1, and contains the 3 quadcodes. Suppose that during time instant t=2, 8 quadcodes are inserted. In such a case, we have a node split. During time instant t=3, a set of 5 quadcodes is deleted. Thus, two nodes of the tree corresponding to time instant t=2 are merged to produce a new node as depicted in the figure.

3 Temporal Window Query Processing

In Spatial Databases and Geographical Information Systems there exists the need for processing a significant number of different spatial queries. For example, nearest neighbor finding, similarity queries, spatial joins of various kinds, window queries, etc. In this section we provide algorithms for the solution of various temporal window queries for evolving regional data. Given a window belonging in the area covered by our images and a time interval the following spatio-temporal queries may be expressed:

3.1 The Strict Containment Query

– Find the black regions that totally fall inside the window (including the ones that touch the window borders from inside) at each time point within the time interval.

In Figure 4 an example of a raster image corresponding to a specific time point, partitioned in quadblocks and a query window are depicted. The Strict Containment window query for this time point would return quadblocks 2 and 4. The algorithm that processes such temporal window queries is as follows:

1. Break the window into maximal sub-quadrants, as if it were a black region represented by a region quadtree.
2. For each of these sub-windows (in order according to the directional code of their North-West corner), compute the smallest and largest directional codes that may appear in the sub-window. The range of these codes includes all the codes (black sub-quadrants) that are strictly included within the sub-window. Perform a respective range search in the B^+tree of the first time point and discover the leaves that either contain such codes or would contain them if they had been inserted. The codes that fall within the above range and appear in these nodes are the black sub-quadrants that are strictly contained within the window for the specific time point.
3. For each leaf discovered in step 2, following the F-pointer at first step and the chain of FC-pointers at second step, discover the leaves that *evolve* from this leaf at the next time point. Discard from further consideration the leaves, the range of which does not intersect with the respective range specified in step 2. The codes that fall within this range and appear in the remaining leaves are the black sub-quadrants that are strictly contained within the sub-window for the specific time point. Proceed to the tree for the next time point by repeating step 3 for each remaining leaf. Stop when the last time point of the time interval is reached.

Note, that when we process the query for a tree of a specific time point, we keep in main memory the nodes discovered for this tree, as well as some of the nodes of the tree of the preceding time point (only those that may lead us to nonaccessed nodes of the tree of the current time point). This holds for all the algorithms presented, except for the one related to the Cover query.

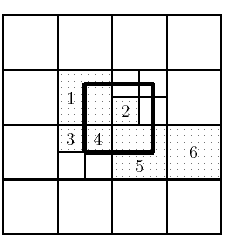


Fig. 4. The quadblocks of a raster image and a query window (thick lines).

3.2 The Border Intersect Query

– Find the black regions that intersect a border of the window (including the ones that touch a border of the window from inside or outside) at each time point within the time interval.

The Border Intersect window query for the time point corresponding to Figure 4 would return quadblocks 1, 3, 4 and 5. The algorithm that processes border intersect is as follows:

1. Create a rectangular strip that is formed by keeping the pixels that make up the border of the query window and the pixels outside the query window that touch its borders. For those sides of the query window that possibly touch the image borders there are no such outside pixels. Therefore, the resulting strip is up to 2 pixels thick. Break the strip into maximal sub-quadrants (of size 2×2, or 1×1).
2. For each of these sub-quadrants (in order according to the directional code of their North-West corner), compute the smallest and largest directional codes that may appear in the sub-quadrant. Then, perform a range search in the B^+tree of the first time point and discover a number of leaves (as in step 2 of the previous algorithm). If such a search returns no quadblocks, search for an ancestor of the sub-quadrant. The quadblocks discovered (if any) intersect the border of the window for the specific time point.
3. For each leaf discovered in step 2, following the F-pointer at first step and the chain of FC-pointers at second step, discover the leaves that *evolve* from this leaf at the next time point. Discard from further consideration the leaves the range of which does not intersect with the respective range specified in step 2. If there are no codes that fall within this range, search for an ancestor of the sub-quadrant corresponding to this range. The codes (quadblocks) discovered (if any) intersect the border of the window for the specific time point. Proceed to the tree for the next time point by repeating step 3 for each remaining leaf. Stop when the last time point of the time interval is reached.

Note, that a search for an ancestor of a quadblock, is a search for the maximum FD code that is smaller than the FD code of the quadblock. If (i) such a code exists, (ii) has a depth D smaller than the quadblock and (iii) the two FD codes coincide in their first D bits, then this FD code corresponds to an ancestor of the quadblock. Such a search can be performed, in most cases, by accessing a very small number of extra disk pages. In more detail, the following sequence of actions is performed. In case the FD under consideration is not the first in its node, we examine the presence of an ancestor in this node. Otherwise, if the previous node is among the nodes that reside in main memory, we examine the presence of an ancestor in this node. If it is not in main memory, but the respective previous node of the preceding tree is in main memory, we use F and possibly FC pointers to reach the previous node for the current tree with very few disk accesses. If, however, none of the above holds, we have to perform a

search in the current tree for the previous node, starting from the root. This search accesses a number of nodes which equals the height of the tree.

Note, also, that the ancestors of a quadblock may be common to a number of subsequent quadblocks (due to the order under which sub-quadrants are treated in step 3). Thus, keeping in a variable the ancestor discovered (if any) for one sub-quadblock may help us to avoid the repetition of the same disk accesses later in the processing of the same time point.

3.3 The Cover Query

– Find out whether or not the window is totally covered by black regions at each time point within the time interval.

The Cover window query returns YES/NO answers. For the time point corresponding to Figure 4, it would return as answer NO. The algorithm that processes this kind of queries is as follows:

1. Break the window into maximal sub-quadrants.
2. For the first of these sub-quadrants (in order according to the directional code of their North-West corner), perform a search in the B^+tree of the first time point and access (discover) the leaf that should contain the related FD code. If the code of the sub-quadrant is present in the leaf, continue. If not, examine the FD codes in the same leaf that are before and after the code of the sub-quadrant (one of them at least exists). If these codes correspond to a sibling or successor of the sub-quadrant mark that the answer for the specific time point will be NO (the window cannot be totally covered). However, continue processing for this time point in order to discover leaves needed for the remaining time points. If the adjacent FD codes do not correspond to siblings or successors, search for an ancestor of this node. If such an ancestor does not exist, mark NO.
3. For the leaf discovered in step 2, following the F-pointer at first step and the chain of FC-pointers at second step, discover the leaves that *evolve* from this leaf at the next time point. Examine these leaves for the presence of the code of step 2, or any of its ancestors or successors (in rare cases, a search from the root of the related tree may be needed in order to examine the presence of an ancestor). According to the nodes discovered, NO may be marked for this time point. Repeat step 3, until you have reached the last image. For those images that have been marked with NO, such that all the images after them have been also marked with NO, the answer is definitely NO. The algorithm does not need to visit them in a subsequent stage and these images are excluded from further consideration. This means that the time interval gets smaller, when we conclude that a number of subsequent images covering its right end have all been marked with NO. Next, repeat step 2 and handle the next unprocessed subquadrant. The algorithm stops when, for every image that has not been excluded, all the sub-quadrants have been processed. For those images that a NO answer has not been marked, the answer for the corresponding time points is YES.

Note the difference of policy from the previous algorithms. In the Cover query we keep in main memory only the nodes related to one quadrant of the current tree, as well as the respective nodes of the preceding tree. It is evident that, we must reserve space for holding the YES/NO answers for all the images in the time interval. This approach is likely to produce NO answers for groups of images and not single images, while it saves us from unnecessary disk accesses.

The Cover window query algorithm can be extended so as to work for partially black windows, where the black percentage exceeds a specified threshold. That is, we could answer a Fuzzy Cover window query of one of the following two forms:

- Find out whether or not the percentage of the window area that is covered by black regions is larger than a given threshold at each time point within the time interval.
- Find out the percentage of the window area that is covered by black regions at each time point within the time interval.

3.4 General Comment for Window Query Algorithms

Alternative naive algorithms for answering the above spatio-temporal queries are easy to devise. These algorithms would perform a suitable range search for all the trees that correspond to the given time interval (starting from the respective roots). This alternative approach would not take into account the "horizontal" pointers that link leaves of different trees and is expected to have significantly worse I/O performance.

All the presented algorithms can be easily transformed to work backwards: by starting from the end of the time interval and by using the B-pointer and BC-pointers.

4 Experiments

We implemented the structure of Overlapping Linear Quadtrees in C++. Note that, in order to maximize overlapping, in our implementation a disk page may host a number of consecutive B^+tree leaves (more details on this B^+tree variation appear in [15]). We performed experiments for page sizes equal to 1K and 2K bytes. For 1K pages, the capacity of internal nodes was 124 keys and the size of each leaf was 1/12 of a page. For 2K pages, the capacity of internal nodes was 252 keys and the size of each leaf was 1/24 of a page. The size of our images was 512×512 pixels and we used the algorithm OPTIMAL_BUILD described in [11] for converting the images from raster to linear FD representation.

At the start, the first image was created and its FD codes were inserted in an empty B^+tree. The codes were inserted one at a time, as they were produced by OPTIMAL_BUILD. Thus, we obtained the result of a typical B^+tree with average storage utilization equal to $\ln 2$. This image represents the last image in a large sequence of overlapped images and its quadcodes were also kept in a main memory compacted binary array ($512 \times 512/8$ bytes). Next, the second

image was created as a modification of the first image and its FD codes were inserted in the second B^+tree, so that the identical subtrees between the two trees overlapped. There was no I/O cost for black quadrants that were identical between the two consecutive images, since, by making use of the main memory compacted array, we were able to sort out the respective identical FD codes.

The first image of each pair was created according to the model of increased image aggregation coefficient [6], $agg(I)$, of an image I. This quantity has been defined and studied in [6] and expresses the coherence of unicolor regions of the image. Starting from a random image with given black/white analogy (an image where each pixel has been colored independently with probabilities that obey this black/white analogy) and using the algorithm presented in [6], an image with the same black/white analogy and higher aggregation (more realistic) can be created.

The second image of each pair was formed by randomly changing the color of a given percentage of the image pixels. Note that the random changing of single pixels is an extreme method of producing evolving images and the results produced by this policy should be seen as very pessimistic. In practice, much better I/O performance is expected for the algorithms presented.

Every experiment was repeated 10 times using a pair of similar images. After the two images were created, windows of sizes equal to 64×64 or to 128×128 pixels were set and each of the three algorithms was executed 10 times for a random window position. In other words, each of the algorithms was run 10×10 = 100 times. Besides, for each window position, the analogous naive algorithms were executed (algorithms that do not make use of horizontal pointers, but perform independent searches through roots). In each run, we kept track of the number of disk reads needed to perform the query.

In the following, various experimental results are depicted, by assuming that the change probability is 2%. In the left part of Figure 5, for the Strict Containment query (processed by making use of horizontal pointers) one can see the number of node accesses as a function of aggregation for various black/white analogies of the first image. The window size is 64×64 pixels and the page size 1K. In the right part of the same figure, for the Strict Containment query one can see the number of node accesses as a function of aggregation for the naive algorithmic approach and the one that uses horizontal pointers. The black percentage is 70% and the page size is again 1K. Results (different plots) for window sizes equal to 64×64 and 128×128 pixels are depicted.

In the left part of Figure 6, another diagram for the Strict Containment query is depicted. Except for the page size which is equal to 2K, the rest of experimental setting is the same as in the previous diagram. In the right part of the same figure an analogous diagram for the Border Intersect query, for page size equal to 1K, is depicted.

In the left part of Figure 7, another diagram for the Border Intersect query is depicted. The experimental setting is identical to the one of the previous diagram, with the exception of the page size which is equal to 2K. The right part of Figure 7 refers to the Cover query: one can see the number of node

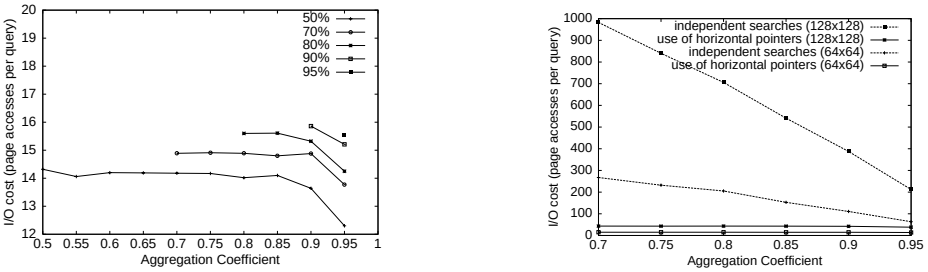


Fig. 5. The I/O efficiency (when page size is 1K) of the Strict Containment query, as a function of aggregation, for various black percentages (left) and for two algorithmic approaches (right).

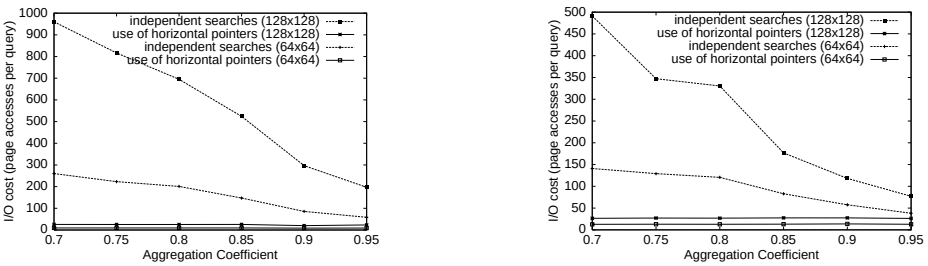


Fig. 6. The I/O efficiency of the Strict Containment auery, for page size of 2K (left) and of the Border Intersect query, for page size of 1K (right), as a function of aggregation.

accesses as a function of aggregation for the naive algorithmic approach and the one that uses horizontal pointers. The window size is 64×64 pixels and the black percentage is 70%. Results (different plots) for page sizes equal to 1K and 2K are depicted.

Note that, since the Cover query is of the YES/NO type and the intelligent exclusion of group of images from further consideration (see subsection 3.3) has been used, the number of node accesses is extremely small. A general remark that can be made for the diagrams in which the naive approach and the approach of Section 3 are compared, is that the use of horizontal pointers leads to significantly higher I/O efficiency for all the three algorithms.

In the future, we plan to perform further experiments for a number of cases. The parameters that may vary in these experiments are: the image size, the disk page size (or the number of leaves fitting in a page), the method of creating the first image, the window size (in relation to the whole image), the black/white analogy for the model based on aggregation, and the percentage of difference in creating the second image of each pair.

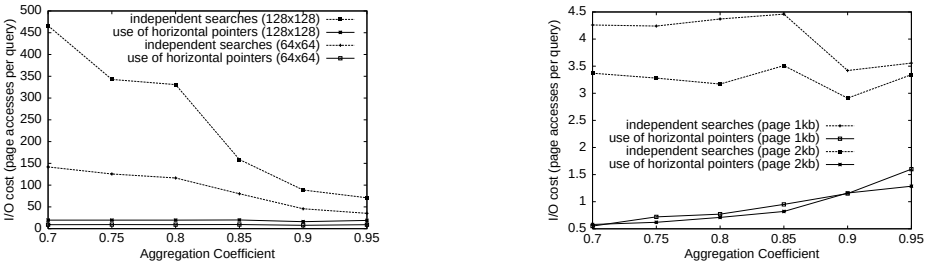


Fig. 7. The I/O efficiency of the Border Intersect query for page size of 2K, (left) and of the Cover query for page sizes of 1K and 2K (right), as a function of aggregation.

5 Conclusions

In this paper, we presented three algorithms for processing spatio-temporal queries in an image database that is organized with Overlapping Linear Quadtrees. More specifically, we presented three temporal window query processing algorithms: strict containment, border intersect and cover. Besides, we presented experiments that we performed for studying the I/O efficiency of these algorithms. The experiments were based on synthetic pairs of evolving images. The first image of each pair was formed according to the model of random images with specified aggregation of black regions [6]. The second image of each pair was formed by random change of pixels, a rather pessimistic method of image changing. In real situations it is expected that the above algorithms will perform even better. Even in this case, our experiments showed that, thanks to the presence of "horizontal" pointers in the leaves of the Overlapping Linear Quadtrees, our algorithms run with very few disk accesses.

In the future, we plan to develop algorithms for other/new spatio-temporal queries that take advantage of Overlapping Linear Quadtrees and study their behavior. Moreover, we plan to investigate the possibility of analyzing the performance of such algorithms.

Acknowledgments

The second author, who is a Post-doctoral Scholar of the State Scholarship Foundation of Greece, would like to thank this foundation for its financial assistance.

References

1. F.W. Burton, M.W. Huntbach and J. Kollias: "Multiple Generation Text Files Using Overlapping Tree Structures", *The Computer Journal*, Vol.28, No.4, pp.414-416, 1985. 87

2. F.W. Burton, J.G. Kollias, V.G. Kollias and D.G. Matsakis: "Implementation of Overlapping B-trees for Time and Space Efficient Representation of Collection of Similar Files", *The Computer Journal*, Vol.33, No.3, pp.279-280, 1990. 87
3. V. Gaede and O. Guenther: "Multidimensional Access Methods", *ACM Computer Surveys*, Vol.30, No.2, pp.170-231, 1998. 85
4. C.S. Jensen, J. Clifford, R. Elmasri, S.K. Gadia, P. Hayes and S. Jajodia (ed.): "A Consensus Glossary of Temporal Database Concepts", *ACM SIGMOD Record*, Vol.23, No.1, pp.52-64, 1994. 85
5. Y. Manolopoulos and G. Kapetanakis: "Overlapping B+trees for Temporal Data", *Proceedings of the 5th Jerusalem Conference on Information Technology (JCIT)*, pp.491-498, Jerusalem, Israel, 1990. Address for downloading: http://delab.csd.auth.gr/publications.html 87
6. Y. Manolopoulos, E. Nardelli, G. Proietti and M. Vassilakopoulos: "On the Generation of Aggregated Random Spatial Regions", *Proceedings of the 4th International Conference on Information and Knowledge Management (CIKM)*, pp.318-325, Washington DC, 1995. 86, 94, 94, 94, 96
7. M. Nascimento and M. Eich: "An Introductory Survey to Indexing Techniques for Temporal Databases", Southern Methodist University, Technical Report, 1995. 86
8. M.A. Nascimento and J.R.O. Silva: "Towards Historical R-trees", *Proceedings of ACM Symposium on Applied Computing (ACM-SAC)*, 1998. 86
9. B. Saltzberg and V. Tsotras: "A Comparison of Access Methods for Time Evolving Data", *ACM Computing Surveys*, to appear. Address for downloading: ftp://ftp.ccs.neu.edu/pub/people/salzberg/tempsurvey.ps.gz. 86
10. H. Samet: "The Design and Analysis of Spatial Data Structures", *Addison-Wesley*, Reading MA, 1990. 85, 87
11. H. Samet: "Applications of Spatial Data Structures", *Addison-Wesley*, Reading MA, 1990. 88, 93
12. Y. Theodoridis, M. Vazirgiannis and T. Sellis: "Spatio-Temporal Indexing for Large Multimedia Applications", *Proceedings of the 3rd IEEE Conference on Multimedia Computing and Systems (ICMCS)*, 1996. 86
13. Y. Theodoridis, T. Sellis, A. Papadopoulos and Y. Manolopoulos: "Specifications for Efficient Indexing in Spatiotemporal Databases", *Proceedings of the 7th Conference on Statistical and Scientific Database Management Systems (SSDBM)*, pp.123-132, Capri, Italy, 1998. 86
14. T. Tzouramanis, Y. Manolopoulos and N. Lorentzos: "Overlapping B+trees - an Implementation of a Transaction Time Access Method", *Data and Knowledge Engineering*, Vol.29, No.3, pp.381-404, 1999. 87
15. T. Tzouramanis, M.Vassilakopoulos and Y. Manolopoulos, "Overlapping Linear Quadtrees: a Spatio-temporal Access Method", *Proceedings of the 6th ACM Symposium on Advances in Geographic Information Systems (ACM-GIS)*, pp.1-7, Bethesda MD, November 1998. 86, 86, 86, 88, 93
16. M. Vassilakopoulos, Y. Manolopoulos and K. Economou: "Overlapping for the Representation of Similar Images", *Image and Vision Computing*, Vol.11, No.5, pp.257-262, 1993. 87
17. M. Vassilakopoulos, Y. Manolopoulos and B. Kroell: "Efficiency Analysis of Overlapped Quadtrees", *Nordic Journal of Computing*, Vol.2, pp.70-84, 1995. 87
18. X. Xu, J. Han, and W. Lu: "RT-tree - an Improved R-tree Index Structure for Spatiotemporal Databases", *Proceedings of the 4th International Symposium on Spatial Data Handling (SDH)*, 1990. 86

On the Correctness of Virtual Partition Algorithm in a Nested Transaction Environment

Sanjay Kumar Madria[1], S. N. Maheshwari[2], and B. Chandra[3]

[1]Department of Computer Science, Purdue University,
West Lafayette, IN-47906
skm@cs.purdue.edu

[2]Department of Computer Science and Engineering,
Indian Institute of Technology, Hauz Khas, New Delhi – 110016, India
snm@cse.iitd.ernet.in

[3]Department of Mathematics Indian Institute of Technology,
Hauz Khas, New Delhi – 110016, India
bchandra@maths.ernet.in

Abstract. In this paper, we model the virtual partition algorithm in a nested transaction environment using I/O automaton model. The formal description is used to construct a complete correctness proof that is based on standard assertional techniques and on a natural correctness condition, and takes advantage of modularity that arises from describing the algorithm as nested transactions. Our presentation and proof treat issues of data replication entirely separately from issues of concurrency control. Moreover, we have identified that virtual partition algorithm can not be proven correct in the sense of Goldman's work [7] on Gifford's Quorum Consensus Algorithm using the serializability theorem defined by Fekete et al.[4]. Thus, we have stated a weaker notion of correctness conditions, which we call reorder serializability theorem.

1 Introduction

By replicating data at multiple sites, the distributed database system can operate even in the case of some site failures or network partitions and therefore, provides increased availability. Replication of the data objects raises the issue of consistency among the replicas. The strongest and simplest notion of consistency is atomicity, which requires the replicas to collectively emulate a single centralized object. To achieve atomicity with replicated data, several methods have been proposed such as write-all/read-one [1], primary-copy [13], majority consensus [14], and quorum consensus ([2],[3],[6]). However, to achieve higher performance, a different notion of consistency, called sequential consistency [8] allows operations to be ordered as long as they remain consistent with the view of individual clients. There are some other systems that provide weaker guarantee of the clients [10] to get better performance. Improving performance by weaker notion of consistency can have more complicated semantics, and may be more difficult to understand and reason the correctness of replicated systems.

Eder et al. (Eds.): ADBIS´99, LNCS 1691, pp.98 -112, 1999.
© Springer-Verlag Berlin Heidelberg 1999

In the virtual partition algorithm of [2] and [3], each logical data object maintains read- and write-quorums [6]. The basic idea here is that each TM maintains a "view" of the data managers (DMs) which the TM (transaction managers) believes it can communicate with. In this algorithm, instead of reading a read-quorum of DMs, the TM can read from any one of the DM in its view having a read-quorum of DMs. Similarly, the TM writes at all DMs in its view containing a write-quorum of DMs. Since the writing is performed at all the DMs in the view and reading can be done at any one of the DM in the view, there is more availability. Also, if a transaction incorrectly believes that a DM in its view has a read-quorum, it may read an old value. It will not be able to discover the new updates until an accessible update is performed in its view. Fortunately, these "correct" read operations can safely be run "as in the past". If a transaction believes that a DM in its view has a write-quorum but in fact does not, then its incorrect view will be discovered at the time write-all is attempted. However, the algorithm does not have enough details to enable complete understanding of its working and to reason about the correctness of transactions.

In [7], Goldman presented a quorum consensus algorithm [7] by incorporating the concept of transaction nesting [12] and transaction aborts. The proof technique in [7] proves that a replicated serial system is same as non-replicated serial system. It then uses the known fact that a concurrent replicated system using the concurrency control algorithm given in [5] is same as non-replicated serial system. This seems to work for a class of replicated serial system but not for all. The virtual partition algorithm [3] has the property that an access in partition A may read a data object that was already modified by a different transaction in partition B. Thus, the virtual partition algorithm allows some read-only transactions to "run as if in the past" and therefore, it does not satisfy the external consistency condition since the apparent serial order has a different sequence when projected on the timestamp order. Hence, it does not satisfy the serial correctness condition as defined in ([4],[7]). That is, the virtual partition algorithm cannot be proved to be "serially correct" [9] as in [7]. Hence, it cannot be combined with algorithms such as those in [4] that provide "one-copy serializable schedules". However, a weaker notion of correctness condition is needed that should allow reordering of transactions to bring the outdated read-only transactions at a point in the past consistent with the ordering of other transactions. Our nested transaction version of the virtual partition algorithm, therefore, can work with any of the algorithms that satisfy the weaker correctness condition. We have stated a weaker notion of correctness which we call reorder serially correct.

In this paper, we have generalized the virtual partition algorithm ([2],[3]) in a framework similar to Goldman's analysis of the quorum consensus algorithm [7] by incorporating the concept of transaction nesting and transaction aborts. In our nested transaction version of the virtual partition algorithm, we have combined view formation and view update protocols in one module that is an improvement over the original algorithm [3].

In this paper, our contribution is restricted to proving that serial replicated system is same as serial non-replicated system as far as user is concerned. However, to prove the correctness of the concurrent replicated system to be equal to serial non-replicated

system, we need to prove that our replication algorithm can be combined with any concurrent algorithm which satisfy the weaker notion of correctness condition. An open problem still remains is to state and prove the reorder serializability theorem based on our reorder serial correctness. Another open problem is to find a concurrent algorithm (that provides the reading of old values) and to prove that it satisfy the weaker notion of correctness condition and the reorder serializability theorem analogous to serializability theorem defined in [9].

2 Nested Transaction System and Correctness

A nested transaction system is modeled by a four-tuple (τ,parent,O,V) where τ is a set of transaction names organized into a tree by the mapping parent : $\tau \rightarrow \tau$ where T_o acts as the root. The set O denotes the set of objects; it partitions the set of accesses, where each partition block contains accesses to the particular objects. V is the set of return values. We can relate two nested transaction systems as follows: A nested transaction system P = (τ_P, $parent_P$, O_P, V_P) is called a structural extension of the nested transaction system Q = (τ_Q, $parent_Q$, O_Q, V_Q) if $\tau_P \supseteq \tau_Q$, $O_P \supseteq O_Q$, respectively, $V_P = V_Q$, and $parent_P$, restricted to τ_Q , = $parent_Q$.

Correctness is defined by first giving a separate specification of permissible serial executions as seen by a user of the system, and then defining how executions of a transaction processing system must relate to this specification. The permissible serial executions for a transaction processing system are defined by introducing the notion of a scheduler, which executes transactions serially. Such serial systems are not constrained by the issues of concurrency control, recovery and transaction aborts.

Serial Correctness: A schedule α of a system is serially correct for a transaction T if its projection on T, $\alpha|T$, is identical to $\beta|T$ for some serial schedule β [4]. In other words, T sees same things in α that it would see in some serial schedule. α is serially correct if it is serially correct for every non-orphan (transactions with no parents), non-access transaction. The serial correctness for all non-orphan transactions implies serial correctness for T_o because the serial scheduler does not have the action ABORT(T_o) so T_o cannot be an orphan.

2.1 Non-Replicated Serial System

In a non-replicated environment, only one copy of each logical data object exists. The non-replicated serial system has, for each logical data object, a corresponding data manager (DM). To perform read and write accesses on a logical data object x, a transaction has to access a DM for x.

The nested transaction system of a non-replicated serial system A = (τ_A,$parent_A$,O_A,V) has the following structure. In the system A, all non-access transactions are user-visible. The nested transaction tree of the system A has access subtransactions at the leaf node and the user-visible transactions at one level above the

access transactions. $parent_A$ is a mapping which gives the parent-child relationships in the transaction tree. O_A is the set of logical data objects that partitions the set of accesses in system A such that each partition contains accesses to one particular data object. All the data objects of system A are user-visible. V_A is a set of return values. Each non-access transaction and data manager is modeled as an I/O automaton. These automata issue request to and receive replies from a serial scheduler which is simply another I/O automaton [4].

3 Virtual Partition Algorithm and Nested Transaction System

The read and write operations of the virtual partition algorithm are modeled by providing two kinds of TMs namely read- and write-TMs. The read and write accesses are accomplished using two types of coordinators namely read- and write-COs. In the algorithm, each transaction maintains its operational view of DMs. To keep the user-visible transactions transparent to the view formed, we have modeled view formation and update protocols as view-coordinator (view-CO) rather than introducing it at the level of TMs. We have kept view-COs at lower levels in the transaction nesting structure so that their invocations and returns are not to be seen by the user-visible transactions. The coordinators (COs) are accommodated in the transaction tree at the intermediate layer between the TMs and the access subtransactions situated at the leaves of the transaction tree. TMs and COs are user-invisible transactions.

The virtual partition algorithm of [2] and [3] in the nested transaction environment works as follows: The set of all data managers (DMs) for the logical data object x models the set of physical replicas for x. DM keeps two types of data items, viz, the user data item {UD} and the control data item {CD} maintained in the form of the tuple <{UD},{CD}>. {UD} and {CD} keep {version-no, value} and {vid, view-no, view}, respectively. The collective form of <{UD},{CD}> is defined over the domain D_x = <{ version-no, value}, {vid, view-no, view }> for x with initial data <{0, I_x}, {I_{vid}, 0, I_{view} }>. The user's data item {UD} can be read and written by subtransactions invoked by any of the COs. The control data item {CD} is written by subtransactions invoked by only view-COs. view, a part of the control data item, is associated with a unique vid (view identifier). vid is used to form a new view. Therefore, each view-CO also maintains a vid. vid of the view-CO is same as of the site where view-CO is initiated. We use o(T) to denote the DM to which T is an access.

The user-visible transactions invoke a TM in order to perform read and write operations. Each TM maintains its operational view of the DMs. When a read-TM performs a logical read of x, it invokes a read-CO that in turn invokes a read access to read one of the DMs for x in its view having some read-quorum. To achieve the non-deterministic nature of the algorithm, the read-CO may invoke more than one read access. The read-CO will accept the value returned by the first committed read access. This value is returned to the user-visible transaction by the read-TM.

To perform a logical write of x with a new value v', a write-TM again invokes a read-CO which reads a DM for x as in case of a logical read. The read returns the value v along with the version-no to the write-TM. The write-TM increases the version-no by

one and invokes a write-CO which further initiates write accesses to write the value v' along with the new version-no at all the DMs in the view of the write-TM containing some write-quorum.

When several invocations to the read- or write-COs fail to read a DM in their view, a read- or write-TM may invoke a view-CO in order to form a new view. The view-CO first generates a vid greater than its present vid and then it initiate a process of reading data components from all the DMs in the system. However, only those DMs having a vid < new vid of the view-CO are eligible to respond. The view-CO keeps in its state all the data components it receives from the DM with highest view-no seen, and the set of the names of the DMs read denoted by d. Once the view-CO is able to read access the minimum number of DMs requires for a read-quorum, it decides about the new view. Then it writes data components at all the DMs in its view or else it could repeat the above procedure to form the new view again.

Formally, we capture the replicated serial system discussed above as a 4-tuple B = (t_B, $parent_B$, O_B, V_B). For each $x \in I$ where I is a set of logical data objects, we define the following: dm(x) is a subset of O_B, acc(x), a subset of the accesses in t_B, is exactly the set of all accesses to objects in dm(x), $tm_r(x)$, $tm_w(x)$ are disjoint subsets of the non-accesses in t_B, and $tm(x) = tm_r(x) \cup tm_w(x)$, co(x) is a subset of the non-access in t_B, config(x) is a legal configuration of dm(x).

The replicas for x will be associated with the members of dm(x) and the logical accesses to x will be managed by the automata associated with the members of tm(x). Also, in the transaction tree, $T \in acc(x)$ iff $parent(T) \in co(x)$, and $T \in co(x)$ iff $parent(T) \in tm(x)$. That is, the accesses to DMs for x are exactly the children of the COs for x. Finally, we require that $dm(x) \cap dm(y) = \Phi$ for any two logical data objects x and y.

The user-visible transactions in B is the set of non-access transactions in t_B that are neither in tm(x) nor in co(x) for all $x \in I$. We refer to accesses in acc(x) for all $x \in I$ as replica accesses and to the remaining accesses (accesses to the non-replicated data objects) in t_B as non-replica accesses.

In system B, each member of dm(x), $tm_r(x)$, $tm_w(x)$ has a corresponding data manager automaton (DM), read-TM automaton, write-TM automaton for x. Each member of co(x) has an associated read-CO, write-CO or view-CO automaton for x.

3.1 Transaction Automata

The TMs and COs, modeled as I/O automata, are specified with the help of the operations given [4]. Here, we give the details of each I/O operation of only write-TM and view-CO. For more details, see [11]. The transition relation, denoted by (s',p,s), have pre and post conditions given separately for each I/O operation p where s' is the state before and s is the state after p. Each state of an automaton has a subset of the following components:

active is a Boolean variable used to control the initiation of CREATE operation, **view-change** is a Boolean variable used to control the initiation of read- and view-COs in the changed view, **read-1** is a Boolean variable used to control the commit of read-COs, **written-1** is a boolean variable used to control the commit of write-Cos. All of the these boolean variables are initially false. **read-req**, **write-req** and **view-req** are the sets of requested read-, write- and view-COs, respectively, **aborted** is the set of aborted transactions, **read-act** and **write-act** are the sets of requested read and write access subtransactions, respectively, **written-2** is the set of data objects updated, **read-2** is the set of data objects read, all of the these sets are initially empty, **data** consists of all the components of the data object. Initially, data is undefined.

We refer the view associated with the transaction T by **view(T)**. Similarly, vid as **vid(T)** and version-no as **version-no(T)**.

Write-TM: A write-TM performs logical write accesses on behalf of the user-visible transactions.

Each state of a write-TM has active, view-change, view-req, read-1, read-req, write-req, written-1, aborted and data as state components used to define pre and post conditions for the following I/O operations:

- **CREATE(T**)
  Precondition: s'.active = false; Postcondition: s.active = true
- **REQUEST-CREATE(T')** where T' is a view-CO
  Precondition: A change in the communication network topology or some sites have failed or previous invocations to the read-COs have failed, i.e., some read-COs $\in$ s'.aborted or some write-COs $\in$ s'.aborted, s'.active = true, T' $\notin$ s'.view-req, s'.view-change = false
  Postcondition: s.view-req = s'.view-req $\cup$ {T'}
- **REPORT-COMMIT(T',v)** where T' is a view-CO
  Postcondition: If s'.view-change = false then s.data = v, s.view-change = true
- **REPORT-ABORT(T',v)** where T' is a view-CO
  Postcondition: no change
- **REQUEST-CREATE(T')** where T' is a read-CO
  Precondition: there is no view-CO $\in$ s'.view-req, s'.active = true, T' $\notin$ s'.read-req, s'.read-1 = false, view(T') = view(T)
  Postcondition: s.read-req = s'.read-req $\cup$ {T'}
- **REQUEST-CREATE(T')** where T' is a read-CO
  Precondition: s'.active = true, T' $\notin$ s'.read-req, s'.read-1 = false, s'.view-change = true, view(T') = view(T)
  Postcondition: s.read-req = s'.read-req $\cup$ {T'}
- **REPORT-COMMIT(T',v)** where T' is a read-CO
  Postcondition: If s'.read-1 = false then s.data = v, s.read-1 = true

- **REPORT-ABORT(T')** where T' is a read-CO
  Postcondition: s.aborted = s'.aborted ∪ {T'}
- **REQUEST-CREATE(T')** where T' is a write-CO and data(T) = d
  Precondition: s'.active = true, view(T') = view(T), T' ∉ s'.write-req, s'.read-1 = true, version-no(T') = s'.data.version-no + 1
  Postcondition: s.write-req = s'.write-req ∪ {T'}
- **REPORT-COMMIT(T',V)** where T' is a write-CO
  Postcondition: s.written-1 = true
- **REPORT-ABORT(T')** where T' is either a write- or view-CO
  Postcondition: s.aborted = s'.aborted ∪ {T'}
- **REQUEST-COMMIT(T,V)**
  Precondition: s'.active = true, v = nil, s'.written-1 = true
  Postcondition: s.active = false

View-CO : The purpose of a view-CO is to form a new view and update the value, version-no, vid, view-no and view at all the DMs in its view. That is, a view-CO models the view formation and update protocols.

A view-CO for x has components active, read-act, write-act, data, read-2 and written-2.

- **CREATE(T)**
  Postcondition: s.active = true
- **REQUEST-CREATE(T')** where T' is a read access
  Precondition: s'.active = true, T' ∉ s'.read-act, vid(T') = vid(T)
  Postcondition: s.read-act = s'.read-act ∪ {T'}
- **REPORT-COMMIT(T',v)** where T' is a read access
  Postcondition: s.read-2 = s'.read-2 ∪ {o(T')}
  If v.version-no > s'.data.version-no then
  { s.data.version-no = v.version-no, s.data.value = v.value }
  If v.view-no > s'.data.view-no then
  { s.data.view-no = v.view-no, s.data.view = v.view }
- **REQUEST-CREATE(T')** where T'is a write access and data(T') = d
  Precondition: s'.active = true, T' ∉ s'.write-act, view(T') = view(T)
  d = <{value(T), version-no(T)},{vid(T), view-no(T), view(T)}>
  Postcondition: s.write-act = s'.write-act ∪ {T'}
- **REPORT-COMMIT(T',V)** where T' is a write access
  Postcondition: s.written-2 = s'.written-2 ∪ {o(T')}
- **REPORT-ABORT(T')** where T' is a write access
  Postcondition: no change

- **REQUEST-COMMIT(T,V)**
  Preconditions: s'.active = true, v = s.data.view, s'.written-2 = view(T)
  p ∈ config(x).r and k ∈ config(x).w such that p and k both ∈ view(T)
  Postcondition: s.active = false

3.2 Data Manager Automaton

Each data manager is modeled as serial object I/O automaton which accepts only read and write access operations. These I/O automata have CREATE(T) as input and REQUEST-COMMIT(T) as output operations. For each data manager automaton associated with the copies of the data object x, the domain of values is D_x.

Each state of the automaton has two components: active which is a Boolean variable and data d is the most recently written value ∈ D_x by the write access subtransaction. Initially, active = false and data = initial data ∈ D_x. Note that v = nil is an element of D_x. The I/O operations along with their pre and post conditions are as follows:

- **CREATE(T)**
  Postcondition: s.active = true
- **REQUEST-COMMIT(T,v)** where T is a read access
  Precondition: {s'.active = true, vid(o(T)) < vid(T), o(T) ∈ view(T), v = s'.data }
  or {s'.active = true, vid(o(T)) = vid(T), v = s'.data }
  Postcondition: s.active = false
- **REQUEST-COMMIT(T,v)** where T is a write access and data(T) = d
  Precondition: s'.active = true, o(T) ∈ view(T), vid(o(T)) = vid(T), v = nil
  Postcondition: s.data = d, s.active = false

4 Replicated Serial System B

We now prove that in the replicated serial system B, each read-TM returns the value written by the previous logical write access in its view. Note that the last logical write in a view may be different than the last write in a system. Thus, a read in a view can return an old value. We first define a reordering of events in a schedule and give some of the definitions that are useful for describing logical accesses to the logical data objects in system B and for giving inductive arguments. Let x be any logical data object.

Reordering of Events: Our nested transaction version of the virtual partition algorithm in a serial environment allows some top level transactions to request the system to allow read-only subtransactions to be run "as in the past". That is, some transactions have the appearance of running in the past and therefore, reading of old values are allowed. This is in accordance with the virtual partition algorithm of [2] and [3]. Therefore, our nested transaction version of virtual partition algorithm does not satisfy the external consistency condition of the serial correctness as defined in [9] (and in

section 2) since the apparent serial order will have a different sequence of operations, for example, when projected on timestamp ordering. Therefore, the virtual partition algorithm cannot be proved to be "serially correct" as in the case of [7]. Thus, we have to define a weaker notion of correctness condition for the systems that permit reading of old values. To make the ordering of such transactions consistent in a serial environment, we move the events of such transactions to a point in the past consistent with the ordering of other transactions. Thus, the user receives responses now that are consistent with them having issued requests in the past and results reported to the user are valid. We formally define reorder as follows: Let α be a schedule of system B. Then, β = reorder(α) is a well-formed schedule of system B where the events are reordered in the sense of above. Later, we will observe that the schedule β satisfies the notion of weaker correctness condition as defined below.

Reorder Serial Correctness: In brief, the weaker notion of correctness, called, ***reorder serial correctness*** can be stated as follows: A sequence β of actions is reorder (not necessarily commutative operations) serially correct for transaction T provided that there is some serial behavior γ such that $\gamma|T$ is a reordering of $\beta|T$. This correctness condition allows reordering of some transactions by moving the events of those transactions in the past by assigning them out of order pseudotime intervals so that it appears that the user requested them earlier. The allowed reordering is in response to user requests and therefore, is acceptable to the user. Thus, the user receives responses that are consistent with the ordering of other transactions such that it seems that they have been issued in the past.

Access sequence: The access sequence of x in β denoted by access(x,β) is defined to be the subsequence of β containing the CREATE and REQUEST-COMMIT operation for the members of $tm_r(x)$ and $tm_w(x)$ [7].

Final-state: The final-state of x after β denoted by final-state(x,β) is defined to be either value(T) if REQUEST-COMMIT(T,v) is the last REQUEST-COMMIT operation for a write-TM in access(x, β), or i_x if no REQUEST-COMMIT operation for a write-TM occurs in access(x, β).

Final-view: The final-view of x after β denoted by final-view(x, β) is defined to be view(T), if

1. REQUEST-COMMIT(T,v) is the last REQUEST-COMMIT operation for a read-TM or write-TM T, and
2. REQUEST-COMMIT(T',v) is the last REQUEST-COMMIT operation for a view-CO T' (child of T) invoked by T in access(x, β), or, i_{view} (initial view) if no REQUEST-COMMIT operation for a view-CO occurs under read- or write-TM in access(x, β).

View-equivalent transactions: Two transactions are said to be view-equivalent if they maintain the same view of the DMs.

Current-version-no: The current-vn(x,β) is defined to be the version-no(T) if REQUEST-COMMIT for T is the last REQUEST-COMMIT operation for a write access to o(T) $\in$ view(T) in β, otherwise current-vn(x,β) = 0.

Current-view-no and Current-vid: same as above.

Lemma 1: Let x be a logical data item in I. Let β be a schedule of the replicated serial system B. If β ends in REQUEST-COMMIT(T,v) with $T \in tm_r(x)$ then v = final-state(x,β). Note that β is a reordered schedule.

To prove the above lemma, we need to establish the following properties involving the control and the user data items. We omitted proofs, see [11] for details.

Property 1: For all DMs $O \in dm(x)$, if d is the data component of O, and d.view-no < current-view-no(x,b) then ∃ some write-quorum q in final-view(x,β) such that for all DMs $O' \in$ final-view(x,β) if d' is the data component of O' then d'.view-no > d.view-no.

Property 2: For all pairs of DMs $O_1, O_2 \in dm(x)$, let d_1 and d_2 be data components of O_1 and O_2 respectively. Then d_1.view-no = d_2.view-no implies that d_1.view = d_2.view and d_1.vid = d_2.vid and view contains read- and write-quorum.

Property 3: There exists a write-quorum $q \in$ config(x).w and $q \in$ final-view(x, β) such that for all DMs $O \in$ final-view(x, β), if d is the data component of O then d .view-no = current-view-no(x, β), d.view = final-view(x, β) and d.vid = current-vid(x,β).

Property 4: There exists a write-quorum $q \in$ config(x).w and $q \in$ final-view(x, β) such that for all DMs $O \in$ final-view(x,β), if d is the data component of O then d.version-no = current-vn(x,β).

Property 5: For all DMs $O \in dm(x)$ in final-view(x,β), if d is the data component of O then d.version-no = current-vn(x, β) implies d.value = final-state(x,β).

To prove the above properties and the lemma, we induct on the length of β.

Base Case: Let β be the empty schedule. Since β is empty, all the components of the user data items as well as of the control data items will hold their respective initial values after β. Thus, all the properties stated above hold trivially. Since β is empty, it does not end in a REQUEST-COMMIT(T,v) with $T \in tm_r(x)$ so the lemma also holds.

Induction Step: Let β = β'ι where β' is a subsequence of the schedule β with REQUEST-COMMIT as the last operation for some $T \in tm(x)$. Let ι be a portion of the schedule after β' starting and ending with the last CREATE and REQUEST-COMMIT operations respectively for some $T_f \in tm(x)$. ι also contains operations corresponding to the subtransactions of T_f. We have access(x,β) = access(x,β')access(x, ι) where access(x, ι) begins with the last CREATE operation in access(x, β). Assume that the lemma holds for β'. By definition, access(x,β) contains only CREATE and REQUEST-COMMIT operations for TMs in tm(x). Also, since β is serial and well formed schedule, access(x,ι) = (CREATE(T_f), REQUEST-COMMIT(T_f,v_f)) for some $T_f \in tm(x)$ and $v_f \in V$.

We note that all accesses in ι to DMs in dm(x) are descendants of T_f. There are two possibilities for T_f, either it is a read- or write-TM. According to our nested transaction

tree, a TM invokes a view-CO to form a new view before performing a logical access or performs a logical access in its associated view. Therefore, in order to show that the induction hypothesis holds for β, T_f should preserve all the properties and the lemma in each of the following four possible cases: T_f is a read-TM and invokes no view-CO, T_f is a read-TM and invokes a view-CO, T_f is write-TM and invokes no view-CO, and T_f is a write-TM and invokes a view-CO.

We know that whenever T_f performs a logical read or write access, it invokes a read-CO first. This implies that in all four cases, when T_f requests to commit in ι, at least one read-CO (a child of T_f) ∈ view(T_f) must commit in ι. Let T' be the first read-CO that commits in response and let ι' be the portion of ι upto and including the commit for T'. Then the following holds.

Observation 1: If s is the state of T' just after a read access transaction commits to T' in ι' then

(a) s.data.version-no and s.data.value contain the current-version-no and the associated final-state of the DM in s.read.
(b) s.data.view-no and s.data.vid contain the current-view-no, current-vid and the associated final-view in s'.read.

Since read-CO T' returns the data read to T_f, the next observation asserts that the current-version-no and the associated final-state, current-view-no, current-vid and final-view in the state s of T_f after β'ι' are same as they were after β'.

Observation 2: The data component of the state of T_f after β'ι' is [{current-vn(x, β'), final-state(x, β')}, {current-vid(x, β'), current-view-no(x, β'), final-view(x, β')}].

Now, for Cases 2 and 3, we will show that T_f does preserve all the properties stated and the lemma. Proofs for the other two cases are omitted.

Case 2: T_f is a read-TM under which a view-CO commits before a read-CO.

Let T' be the first view-CO that commits to T_f and let ι' be a portion of ι upto and including that commit. Then the following holds.

Observation 3: If s is the state of T' just after a read access commits to T' in ι', then s.data.view-no contains the highest view-no and the associated view among DMs in s.read. Similar statements hold for the version-no and the associated value.

After a new view is formed, a view-CO T' invokes write accesses T in ι with data(T) to update all the DMs in view (T'). The following observation implies that the data component after β of T has current-version-no, and the associated final-state, current-view-no, current-vid and final-view and also the current-version-no and current-view-no after β are greater than what they were after β'.

Observation 4: If T is a write access invoked by the view-CO for x in ι then data(T) = [{current-vn(x,β), final-state(x,β)}, {current-vid, current-view-no(x,β), final-view(x,β)}] and current-view-no(x,β) > current-view-no(x, β'), current-vid(x, β) > current-vid(x, β') and final-view(x, β) will be the new final-view after β'.

We now show with the help of Observations 1 to 4, that all the properties and the Lemma 1 holds in Case 2. Note that read-TM T_f cannot request to commit until at least one of its read-CO commits after a committed view-CO by the definition of T_f. Let T_r be the first read-CO that commits to T_f and let ι" be the portion of ι upto and including the commit of T_r. We claim that all the properties of the induction hypothesis hold after βι". If ι" contains one or more commits of view-COs, then each view-CO T_v cannot commit until it has received commit operation for all write accesses initiated by T_v in view(T_v). We now show that Property 1 holds after βι". Consider all the DMs that have view-no = current-view-no(x,β') after βι". They must have view = final-view(x,β') (Observation 4 and Property 1 of the induction hypothesis). Observation 4 implies that view-no(T_v) = current-view-no(x, β') + 1 and view(T_v) = new final-view after β'. Therefore, Property 1 holds after βι".

We also know by Observation 4 that all write accesses for x in ι" with view-no ≠ ^ (Note that ^ is a special place holder symbol) have view-no greater than any view-no for x in β'. Furthermore, since all such write accesses have the same view-no, the view and vid are also the same. Therefore, Property 2 continues to hold. Since T_r is a read-CO, it cannot request to commit until it has received commit operation for read access in a view(T_r) having a read-quorum. Since T_r is a read-CO and by Observation 2, Properties 3, 4 and 5 continue to hold after βι". Thus, claim is true and hence, all the properties hold after βι".

By Observation 4, any view-CO that may commit in ι after ι" propagate new vid, view-no and view. So T_f preserves all the properties of the induction hypothesis. Any read-CO that executes in ι after ι" cannot change the data components of the DMs. Since T_f is a read-TM, final-state(x, β) = final-state(x, β') by Observation 2. Thus, the Lemma 1 holds in case 2.

Case 3: T_f is a write-TM and invokes no view-CO.

Since T_f is a write-TM, it invokes write-COs for x to write at all the DMs in the view of T_f then the following holds.

Observation 5: All write-COs T_w for x invoked in ι have version-no(T_w) = current-vn(x, β') + 1, value(T_w) = value(T_f), view(T) = final-view(x, β').

A write-CO in turn invokes write access transactions T. The data component (i.e., current-version-no and the associated final-sate) associated with T after β is as given below. Also, the current-version-no after β is greater than current-version-no after β'.

Observation 6: If T is a write access invoked in ι then data(T) = [{ current-vn(x,b), final-state(x,b) }, {^,^,^}] and current-vn(x,β) > current-vn(x,β').

By Observation 6, the vid, view-no and view in the states of DMs for x are not changed during ι. Therefore, Properties 1, 2 and 3 hold after β. Since T_f is a write-TM, it cannot request to commit until at least one of its write-CO ∈ view(T_f) commits. Let T_w be the first write-CO that commits to T_f and let ι" be the portion of ι upto and including the commit for a write access in final-view(x,β'). By Observation 5,

view(T_w) = final-view(x,β') which equals final-view(x,β). Therefore, by Observation 6, Properties 4 and 5 hold after β'ι".

We now show that all properties hold after β'ι. By Observation 6, any write-CO that may execute in ι after ι" merely propagates the new value and the version-no and any read-CO that may execute in ι after ι" cannot change the values at the DMs since they do not invoke write accesses. Therefore, all the properties hold after β'ι = β. Since T_f is not a read-TM so the Lemma 1 holds vacuously.

5 Correctness

The correctness proof of the algorithm has a two tier structure. First, we will observe that the replicated serial system B under the virtual partition algorithm in a nested transaction environment is a structural extension of the non-replicated serial system A, and system B is serially correct with respect to the system A. That is, the user-visible transactions cannot distinguish between the systems A and B. In effect, the user-visible transactions in system B will have the same executions as the corresponding user-visible transactions in system A and the values observed by them are same in both A and B.

5.1 Correctness of Replicated Serial System B

Notice that in both systems A and B, the transaction tree structures upto the level of user-visible transactions is the same. In the non-replicated serial system A, all non-access transactions are user-visible transactions. Also, if T is a user-visible transaction in B then there is a corresponding user-visible transaction in A. Each non-access transaction of A is represented by the same automaton in both A and B. If T is a TM in system B then there is a corresponding access transaction in system A. There are no transactions in A which correspond to the coordinators and access transactions of system B. The objects that correspond to tm(x) in B will be the single copy read-write objects o(x) in system A. The user-visible objects in both systems, modeled by the same automata, are the same. By comparing these two systems, it follows that systems A and B are such that the sets ι_B and O_B contain τ_A and O_A, respectively, $V_B = V_A$, and $parent_B$, restricted to $\tau_A = parent_A$. That is, system B is a structural extension of the system A. The following two properties are satisfied by systems A and B:

1. The corresponding user-visible transactions in both systems A and B are modeled by the same corresponding automata and therefore, they will have the same schedule in the schedules of systems A and B.
2. The user-visible data objects are modeled by the same automata in both systems A and B. Therefore, the states of all user-visible data objects will be same in some executions of both A and B. In other words, the sequence of REQUEST-COMMIT of write access subtransactions on the user-visible data objects will be the same in some schedules of both systems A and B. That is, the user-visible data objects will have the same schedules in both A and B.

Now consider a schedule β of system B consisting of operations corresponding to user-visible transactions and user-invisible transactions namely, TMs, COs and access subtransactions. We construct a sequence of operations α of system A by

i. removing from β all the REQUEST-CREATE(T), CREATE(T), REQUEST-COMMIT(T,v), REPORT-COMMIT(T,v) and REPORT-ABORT(T) operations for all transactions T in acc(x), and CO(x) for all $x \in I$,
ii. by interpreting T, a user-visible transaction in B, to stand for the corresponding user-visible transaction in A, and
iii. replacing read-/write-TM operations by equivalent read/write access operations.

Note that the equivalent user-visible transactions as well as user-invisible data objects in the constructed sequence of operations α of A and in the schedule β of B have the same order. Therefore, to establish the correctness of the serial system B with respect to system A, we have to only show that α is indeed a schedule of the non-replicated serial system A, and that the read accesses in α returns the same value as those returned by the corresponding read-TMs in β. Note that β is a reordered schedule. The proof of this is by induction on the length of β. We have omitted details.

6 Open Problem

In our discussion, to complete the correctness, we need to prove that the concurrent nested transaction replicated system under the virtual partition algorithm is also correct.

To prove the correctness of concurrent part of the algorithm in [7], it has been stated that any “correct” concurrency control algorithm (that satisfy the serializability theorem stated in [9]) can be combined with their replication algorithm to yield a correct system. In general, any concurrency control algorithm that provides atomicity (as defined in [9]) at the levels of replicas may work with their replication algorithm and will produce a correct system. However, we can not state this with respect to our replication algorithm as the concurrency control algorithms in [4] satisfy the “serial correctness” as defined there. However, as stated, in our replication algorithm, a read may miss a preceding write in another view, thus can produce an execution which is not atomic. Thus, our replication algorithm can not be proven serially correct using the definition of [9]. Thus, we have defined a reorder serial correctness. Therefore, to prove the correctness of a concurrent system combined with our replication algorithm, we have to find a concurrency control algorithm which allows some read-only transactions to run "as if in the past" and then we should be able to prove the correctness of that algorithm using reorder serial correctness. We also need to state and prove a reorder serializability theorem based on reorder serial correctness which should conclude that a sequence of operations is reorder serializable for the root transaction T_0. Thus, these are still remain open problems. Once having proved that, we can also simply state as in [7] that our replication algorithm can be combined with any concurrency control algorithm that satisfies the reorder serial correctness.

7 Conclusions

In this paper, we have modeled the virtual partition algorithm in a nested transaction environment using I/O automaton model. We have proved that replicated serial system under virtual partition algorithm is same as non-replicated serial system as far user is concerned. Our proof treats issues of replication separately from concurrency control algorithm. More importantly, we have identified that not all class of replication algorithms can be proven "serially correct" in the sense of [7,9]. We have defined a new correctness condition called reorder serial correctness. For the future work, we are working on the proof of correctness of the reorder serializability theoem.

8 References

[1] Bernstein, P., Hadzilacos, V. and Goodman, N., Concurrency Control and Recovery in Database Systems, Addision-Wesley, 1987.

[2] El Abbadi, A., Skeen, D. and Cristian, F., An Efficient Fault-Tolerant Protocol for Replicated Data Management, in Proceedings of the 4th ACM Symposium on Principles of Database Systems, pp. 215-228, Portland, Oregon, 1985.

[3] El Abbadi, A. and Tough, S., Availability in Partitioned Replicated Databases, in Proceedings of the 5th ACM symposium on Principles of Database Systems, pp. 240-251, Cambridge, MA, 1986.

[4] Fekete, A., Lynch, N., Merrit, M. and Whiel, W. E., Atomic Transactions, Morgan-Kaufmann, USA, 1994.

[5] Fekete, A., Lynch, N., Merrit, M. and Whiel, W. E., Nested Transactions and Read/Write Locking, in Proceedings of the 6th ACM Symposium on Principles of Database Systems, pp. 97-111, San Diego, CA, 1987.

[6] Gifford, D., Weighted Voting for Replicated Data, in Proceedings of the 7th Symposium on Operating Systems Principles, pp. 150-159, Pacific Grove, CA, 1979.

[7] Goldman, K. and Lynch, N., Nested Transactions and Quorum Consensus, ACM Transactions on Database Systems, Vol. 19, No. 4, pp. 537-585, 1994.

[8] Lee, J.K., Precision Locking for Nested Transaction Systems, in proceedings of Second International Conference on Information and Knowledge Management (CIKM'93), USA, 1993.

[9] Lynch, N. and Merrit, M., Introduction to the Theory of Nested Transactions, Theoretical Computer Science, Vol. 62, pp. 123-185, 1988.

[10] Oki, B., and Liskov, B., Viewstamp Replication : A New Primary Copy Method to Support Highly-available Distributed Systems, in Proceedings of the 7th ACM Symposium on Principles of Distributed Computing, Aug.1988.

[11] Madria, S. K., Concurrency Control and Recovery Algorithms in Nested Transaction Environment and Their Proofs of Correctness, Ph.D. Thesis, Department of Mathematics, Indian Institute of Technology, Delhi, India, 1995.

[12] Moss, J.E.B., Nested Transactions: An Approach to Reliable Distributed Computing, MIT press, Cambridge, MA, 1985.

[13] Stonebraker, M., Concurrency Control and Consistency of Multiple Copies of Data in Distributed INGRES, IEEE Transaction on Software Engineering, 5(3):188-194, May 1979.

[14] Thomas, R., A Majority Consensus Approach to Concurrency Control for Multiple Copy Databases, ACM Transaction on Database Systems, 4(2):180-209, June, 1979.

Computing Rules for Detecting Contradictory Transaction Termination Dependencies

Kerstin Schwarz, Can Türker, and Gunter Saake

Otto-von-Guericke-Universität Magdeburg, ITI
Postfach 4120, D–39016 Magdeburg
{schwarz,tuerker,saake}@iti.cs.uni-magdeburg.de

Abstract. Complex applications consist of a large set of interrelated transactions. Often, it is very difficult for the application/transaction designer to get a grasp of the transitive relationships among the transactions of a complex application. In this paper, we discuss transitive termination dependencies under consideration of transaction compensation and present an algorithm to derive a set of rules for reasoning about transitive dependencies. These rules may help the designer in understanding the entire semantics of a complex application and detecting contradictory dependency specifications.

1 Introduction

The dynamics of complex applications can be modeled by collections of related transactions. Examples for complex applications are business processes, CSCW applications or design transactions. In such applications there are different kinds of dependencies among the related transactions, e.g. termination dependencies. Often, it is very difficult for the application/transaction designer to get a grasp of the transitive relationships among the transactions of a complex application (which is necessary to understand the complete semantics of the modeled application). The concept of *transaction closure* [9] provides a common framework for modeling advanced, complex transactions/activities. A transaction closure comprises of a set of transactions which are transitively initiated by the same (root) transaction. In contrast to nested transactions, a child transaction in a transaction closure may survive the termination of its parent transaction — a case which is needed for example for long-during activities [4], transactional workflows [11], or active databases [2].

Termination dependencies play a central role in transaction closures. A termination dependency is a constraint on the possible combinations of the termination events (commit/abort) of related transactions. Whereas termination dependencies between parent and direct child transactions have to be explicitly specified, the dependencies for transitively related transactions can be computed based on the direct dependencies. As a rule, transitive dependencies have to be compatible with the specified direct dependencies. Using transitive dependencies we are able to detect contradictory dependency specifications and conclude how arbitrary transactions are interrelated.

Eder et al. (Eds.): ADBIS'99, LNCS 1691, pp. 113–127, 1999.
© Springer-Verlag Berlin Heidelberg 1999

In this paper, we present an algorithm to evaluate transitivity rules for termination dependencies. The algorithm also considers the aspect of transaction compensation and its impact on termination dependencies. Other approaches like [7,1,10,6] deal with the enforcement of transaction dependencies by scheduling incoming events. Our approach focus on the design of transactions and the dependencies among them. We are able to check the correctness of a design before execution. In particular, we can detect inconsistent and redundant dependencies. Our correctness criteria forbids dependency combinations that lead to transactions which never can be executed successfully. In contrast the other approaches, we explicitly consider transaction compensation. The whole framework provides the basis for a transaction design and analyzing tool. Such a tool can help to understand the entire semantics of a complex application, to detect contradictory dependency specifications, and, thus, to support the design of better and more efficient applications.

The paper is organized as follows: Section 2 introduces the notion of transaction closure. In Section 3, we consider the transitivity of termination dependencies and develop an algorithm for deriving all valid combinations of dependencies for a given transitive dependency. The application of transaction closures, especially the detection of contradictory dependencies, is presented in Section 4.

2 Transaction Closures and Termination Dependencies

Traditionally, a *transaction* is an execution unit consisting of a set of database operations. A transaction t_i is started by invoking the primitive *begin* (b_{t_i}) and is terminated by either *commit* (c_{t_i}) or *abort* (a_{t_i}). These primitives are termed as *significant events* [3]. A set of transactions with dependencies among them can be considered as *transaction closure* [9]. Each transaction closure consist of exactly one root transaction and a set of non-root transactions which has exactly one parent transaction. This structure is acyclic. The effects of transactions on other transactions are described by dependencies which are constraints on possible histories. We distinguish between *termination*, *execution*, and *object visibility dependencies*. In this paper our focus lies on termination dependencies.

Constraints on the occurrence of the significant termination events commit and abort leads to different termination dependencies. In case of two transactions t_i and t_j there are four possible combinations of termination events: both transactions abort, one transaction commits whereas the other one aborts, and both transactions commit. These termination event combinations may be valid in any order (denoted by ✓) or are not valid (denoted by —).

As depicted in Table 1, we identify five dependencies as reasonable according to real-world application semantics. The termination dependency between t_i and t_j is called *vital-dependent*, denoted as $vital\_dep(t_i, t_j)$, if the abort of transaction t_i leads to the abort of t_j and vice versa. Thus, either both transactions commit together or both abort. The *vital* dependency between t_i and t_j, denoted as $vital(t_i, t_j)$, concerns the case where the abort of transaction t_i leads to the abort of t_j. The inverse relationship, i.e. the fact that transaction t_i has to

abort if t_j aborts, is denoted as $dep(t_i, t_j)$. Obviously, $vital(t_i, t_j) \equiv dep(t_j, t_i)$ holds. An *exclusive* dependency $exc(t_i, t_j)$ demands that the transactions t_i and t_j are not allowed to commit together. Finally, the dependency $indep(t_i, t_j)$ concerns the case where each combination of transaction termination events is valid. Therefore, the involved transactions are called *independent*.

t_i	t_j	$vital\_dep(t_i, t_j)$	$vital(t_i, t_j)$	$dep(t_i, t_j)$	$exc(t_i, t_j)$	$indep(t_i, t_j)$
a_{t_i}	a_{t_j}	✓	✓	✓	✓	✓
a_{t_i}	c_{t_j}	—	—	✓	✓	✓
c_{t_i}	a_{t_j}	—	✓	—	✓	✓
c_{t_i}	c_{t_j}	✓	✓	✓	—	✓

Table 1. Termination Dependencies between two Transactions t_i and t_j

Termination dependencies can be relaxed when transaction compensation is supported. Transaction compensation [5] is the ability to semantically undo the effects of a (committed) transaction t_i by executing a *compensating* transaction, denoted as cmp_{t_i}. If there exists such a compensating transaction cmp_{t_i} for a transaction t_i, t_i is called *compensatable*. The compensatable property allows to commit a transaction without knowing the termination result of a dependent transaction. For instance, in case of $vital(t_i, t_j)$ the transaction t_j can commit before transaction t_i terminates, if t_j is compensatable. In case t_i aborts, t_j is "semantically aborted" by executing the compensating transaction cmp_{t_j}.

We now consider the termination dependencies with respect to the fact that the related transactions may be compensatable. For the symmetrical dependencies $vital\_dep(t_i, t_j)$ and $exc(t_i, t_j)$ we have to distinguish the cases where only one or both of the related transaction are compensatable. The symbols ◀, ▶, and ⧫ are used to reflect which one of the related transactions is compensatable. For example, in case of $vital\_dep(t_i, t_j)$ we distinguish the following cases:

$$vital\_dep_{◀}(t_i, t_j) \text{ — only } t_i \text{ is compensatable}$$
$$vital\_dep_{▶}(t_i, t_j) \text{ — only } t_j \text{ is compensatable}$$
$$vital\_dep_{⧫}(t_i, t_j) \text{ — } t_i \text{ and } t_j \text{ are compensatable}$$

Considering the termination events, a compensating transaction requires that the commit of the compensatable transaction precedes the abort of the third transaction. In this case, the compensating transaction has to commit. As illustrated in Table 2, we can now relax the basic termination dependencies as follows. In case t_i and t_j are compensatable, all disallowed fields (except the two fields in $vital\_dep_{◀}(t_i, t_j)$ and $vital\_dep_{▶}(t_i, t_j)$, respectively) are relaxed by a constraint on the termination order of the related transactions or by the requirement that there must be a committed compensating transaction (compare Table 1 and Table 2). In case of the exclusive dependency we do not need the ordering requirement. If both transactions commit, then one of the compensating

transaction has to commit, too. For the dependency $vital_{\blacktriangleright}(t_i, t_j)$ only transaction t_j is relevant to be compensatable. In case transaction t_i is compensatable and t_i commits, the results of t_i do not need to be compensated because both termination event combinations (c_{t_i}, a_{t_j}) and (c_{t_i}, c_{t_j}) are valid. Thus, there is no constraint on the execution of t_i which could be relaxed by compensation. The same holds for $dep_{\blacktriangleleft}(t_i, t_j)$ and a compensatable transaction t_j.

t_i	t_j	$vital\_dep_{\blacklozenge}(t_i, t_j)$	$vital\_dep_{\blacktriangleleft}(t_i, t_j)$	$vital\_dep_{\blacktriangleright}(t_i, t_j)$	$vital_{\blacktriangleright}(t_i, t_j)$
a_{t_i}	a_{t_j}	✓	✓	✓	✓
a_{t_i}	c_{t_j}	$c_{t_j} \rightarrow a_{t_i} \wedge c_{cmp_{t_j}}$	—	$c_{t_j} \rightarrow a_{t_i} \wedge c_{cmp_{t_j}}$	$c_{t_j} \rightarrow a_{t_i} \wedge c_{cmp_{t_j}}$
c_{t_i}	a_{t_j}	$c_{t_i} \rightarrow a_{t_j} \wedge c_{cmp_{t_i}}$	$c_{t_i} \rightarrow a_{t_j} \wedge c_{cmp_{t_i}}$	—	✓
c_{t_i}	c_{t_j}	✓	✓	✓	✓

t_i	t_j	$exc_{\blacklozenge}(t_i, t_j)$	$exc_{\blacktriangleleft}(t_i, t_j)$	$exc_{\blacktriangleright}(t_i, t_j)$	$dep_{\blacktriangleleft}(t_i, t_j)$
a_{t_i}	a_{t_j}	✓	✓	✓	✓
a_{t_i}	c_{t_j}	✓	✓	✓	✓
c_{t_i}	a_{t_j}	✓	✓	✓	$c_{t_i} \rightarrow a_{t_j} \wedge c_{cmp_{t_i}}$
c_{t_i}	c_{t_j}	$c_{cmp_{t_i}} \vee c_{cmp_{t_j}}$	$c_{cmp_{t_i}}$	$c_{cmp_{t_j}}$	✓

Table 2. Relaxed Termination Dependencies

3 Transitivity of Termination Dependencies

A termination dependency may have a transitive influence on other transactions of the transaction closure. Suppose, X is a termination dependency between the transactions t_i and t_k and Y is a dependency between the transactions t_k and t_j, then the combination of these two dependencies leads to a transitive dependency Z between the transactions t_i and t_j; i.e., $X(t_i, t_k) \wedge Y(t_k, t_j) \Rightarrow Z(t_i, t_j)$.

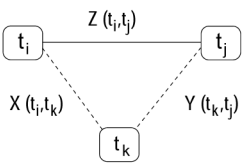


Fig. 1. Example of a Transitive Dependency

Our idea is now to find all valid combinations of dependencies X and Y for a given transitive dependency Z (see Figure 1). For that, we analyze the basic relationships between the dependencies introduced so far. Based on the results how two dependencies can be combined, we present a method for deriving all valid combinations of dependencies X and Y for a transitive dependency Z.

3.1 Foundations of Deriving Transitive Dependencies

Before introducing the method to derive all valid combinations of dependencies for a given transitive dependency, we have to investigate the following basic formula $X(t_i, t_k) \wedge Y(t_k, t_j) \Rightarrow Z(t_i, t_j)$ in more detail. Concerning the possible termination event combinations, we distinguish five possibilities for the application of the dependencies X, Y, and Z on a pair of transactions (denoted by the following symbols):

- $\ominus$ The combination of termination events is *not valid*.
- $\oplus$ The termination event combination is *valid* independent of the termination order and compensation properties.
- $\blacklozenge$ The combination of termination events is valid under the restriction that both transactions are compensatable.
- $\blacktriangleleft$, $\blacktriangleright$ In this case only one of the related transactions is compensatable. The direction of the triangle indicates the transaction which has to be compensatable.

The usage of these symbols is illustrated in Table 3 (cf. Table 2).

t_i	t_j	$vital\_dep_{\blacklozenge}(t_i, t_j)$	$vital\_dep_{\blacktriangleright}(t_i, t_j)$	$vital_{\blacktriangleright}(t_i, t_j)$	$exc_{\blacklozenge}(t_i, t_j)$	$exc_{\blacktriangleleft}(t_i, t_j)$
a_{t_i}	a_{t_j}	$\oplus$	$\oplus$	$\oplus$	$\oplus$	$\oplus$
a_{t_i}	c_{t_j}	$\blacktriangleright$	$\blacktriangleright$	$\blacktriangleright$	$\oplus$	$\oplus$
c_{t_i}	a_{t_j}	$\blacktriangleleft$	$\ominus$	$\oplus$	$\oplus$	$\oplus$
c_{t_i}	c_{t_j}	$\oplus$	$\oplus$	$\oplus$	$\blacklozenge$	$\blacktriangleleft$

Table 3. Graphical Representation of Dependencies with Compensation

The transactions t_i, t_j, and t_k of the dependencies $X(t_i, t_k)$, $Y(t_k, t_j)$, and $Z(t_i, t_j)$ may either abort or commit. Thus, there are four possible termination event combinations for each dependency and eight possible termination event combinations for these transactions. From this we derive the following formulas for evaluating the transitive dependency Z:

$$(x_{(a,a)} \wedge y_{(a,a)}) \vee (x_{(a,c)} \wedge y_{(c,a)}) \Rightarrow z_{(a,a)} \quad (1)$$

$$(x_{(a,a)} \wedge y_{(a,c)}) \vee (x_{(a,c)} \wedge y_{(c,c)}) \Rightarrow z_{(a,c)} \quad (2)$$

$$(x_{(c,a)} \wedge y_{(a,a)}) \vee (x_{(c,c)} \wedge y_{(c,a)}) \Rightarrow z_{(c,a)} \quad (3)$$

$$(x_{(c,a)} \wedge y_{(a,c)}) \vee (x_{(c,c)} \wedge y_{(c,c)}) \Rightarrow z_{(c,c)} \quad (4)$$

The conjunction ($\wedge$) and disjunction ($\vee$) of two dependencies x and y are defined in Table 4. From Table 4 follows that $\ominus$ is stronger than ($\blacklozenge, \blacktriangleleft, \blacktriangleright$), whereas ($\blacklozenge, \blacktriangleleft, \blacktriangleright$) are stronger than $\oplus$. However, for combinations with ($\blacklozenge, \blacktriangleleft, \blacktriangleright$) we have to consider the direction of the dependencies. The $\blacklozenge$-symbol can be seen as a combination of $\blacktriangleleft$ and $\blacktriangleright$. For the $\wedge$-connectivity we have to find a way along the two triangles to get a triangle as result, e.g. $\blacktriangleright$ and $\blacktriangleright$ leads to $\blacktriangleright$, whereas $\blacktriangleright$ and $\blacktriangleleft$ leads to $\ominus$. If one of the elements is $\oplus$, then the way along the symbols

x	y	$(x \wedge y)$	$(x \vee y)$
$\ominus$	$\oplus$	$\ominus$	$\oplus$
$\ominus$	$\blacktriangleright$	$\ominus$	$\blacktriangleright$
$\ominus$	$\blacktriangleleft$	$\ominus$	$\blacktriangleleft$
$\ominus$	$\blacklozenge$	$\ominus$	$\blacklozenge$
$\ominus$	$\ominus$	$\ominus$	$\ominus$
$\oplus$	$\oplus$	$\oplus$	$\oplus$
$\oplus$	$\blacktriangleright$	$\blacktriangleright$	$\oplus$
$\oplus$	$\blacktriangleleft$	$\ominus$	$\oplus$
$\oplus$	$\blacklozenge$	$\blacktriangleright$	$\oplus$
$\oplus$	$\ominus$	$\ominus$	$\oplus$

x	y	$(x \wedge y)$	$(x \vee y)$
$\blacktriangleleft$	$\oplus$	$\blacktriangleleft$	$\oplus$
$\blacktriangleleft$	$\blacktriangleright$	$\blacklozenge$	$\blacklozenge$
$\blacktriangleleft$	$\blacktriangleleft$	$\blacktriangleleft$	$\blacktriangleleft$
$\blacktriangleleft$	$\blacklozenge$	$\blacklozenge$	$\blacklozenge$
$\blacktriangleleft$	$\ominus$	$\ominus$	$\blacktriangleleft$
$\blacktriangleright$	$\oplus$	$\ominus$	$\oplus$
$\blacktriangleright$	$\blacktriangleright$	$\blacktriangleright$	$\blacktriangleright$
$\blacktriangleright$	$\blacktriangleleft$	$\ominus$	$\blacklozenge$
$\blacktriangleright$	$\blacklozenge$	$\blacktriangleright$	$\blacklozenge$
$\blacktriangleright$	$\ominus$	$\ominus$	$\blacktriangleright$

x	y	$(x \wedge y)$	$(x \vee y)$
$\blacklozenge$	$\oplus$	$\blacktriangleleft$	$\oplus$
$\blacklozenge$	$\blacktriangleright$	$\blacklozenge$	$\blacklozenge$
$\blacklozenge$	$\blacktriangleleft$	$\blacktriangleleft$	$\blacklozenge$
$\blacklozenge$	$\blacklozenge$	$\blacklozenge$	$\blacklozenge$
$\blacklozenge$	$\ominus$	$\ominus$	$\blacklozenge$

Table 4. Conjunction and Disjunction of x and y

has to end at the top of a triangle, e.g. $\oplus$ and $\blacktriangleright$ leads to $\blacktriangleright$, whereas $\oplus$ and $\blacktriangleleft$ leads to $\ominus$.

The reason for these results is that the connection of two symbols by an $\wedge$-operator is a combination of termination events of t_i, t_k, and t_j. The first symbol represents the connection between the termination events of t_i and t_k and the second of t_k and t_j. Obviously, the termination event of t_k has to be the same in both cases. The combination of $\blacktriangleright$ and $\blacktriangleleft$ (t_k is compensatable) leads to $\ominus$, because neither t_i nor t_j are compensatable in this case and we have to set a stronger constraint on the termination event combination than the possibility of compensation represented by a triangle.

We are able to derive for each pair of termination dependencies $X(t_i, t_k)$ and $Y(t_k, t_j)$ the corresponding transitive termination dependency $Z(t_i, t_j)$ by initializing the formulas (1)–(4) with the symbols of X and Y and applying the rules represented in Table 4. For example, the dependency $vital\_dep_{\blacktriangleleft}(t_i, t_k)$ $(\oplus, \ominus, \blacktriangleleft, \oplus)$ and the dependency $vital\_dep_{\blacktriangleright}(t_k, t_j)$ $(\oplus, \blacktriangleright, \ominus, \oplus)$ can be derived to $vital\_dep_{\blacklozenge}(t_i, t_j)$ $(\oplus, \blacktriangleright, \blacktriangleleft, \oplus)$ as follows:

$$(\oplus \wedge \oplus) \vee (\ominus \wedge \ominus) \Rightarrow \oplus$$
$$(\oplus \wedge \blacktriangleright) \vee (\ominus \wedge \oplus) \Rightarrow \blacktriangleright$$
$$(\blacktriangleleft \wedge \oplus) \vee (\oplus \wedge \ominus) \Rightarrow \blacktriangleleft$$
$$(\blacktriangleleft \wedge \blacktriangleright) \vee (\oplus \wedge \oplus) \Rightarrow \oplus$$

3.2 Computing Valid Dependency Combinations

In this subsection we start with a given dependency $Z(t_i, t_j)$, e.g. dependency $vital\_dep_{\blacklozenge}(t_i, t_j)$ with $z_{(a,a)} = \oplus$, $z_{(a,c)} = \blacktriangleright$, $z_{(c,a)} = \blacktriangleleft$, and $z_{(c,c)} = \oplus$. The following general form of the formulas (1)–(4) build the basis for the evaluation of the values of x_1, y_1, x_2, and y_2 of the dependencies X and Y with a given symbol z_m, e.g. $z_{(a,a)} = \oplus$:

$$((x_1 \wedge y_1) \vee (x_2 \wedge y_2)) \Rightarrow z_m$$

We try to find the minimal set of values (symbols) for x_1, y_1, x_2, and y_2 which fulfill the formula above. For each symbol the transitive event combination z_m is separately computed in a step of the following algorithm:

```
procedure Deriving_Dependency_Combinations
  // Input:  Transitive Dependency Z
  // Output: Set of Dependency Combinations X and Y which imply Z
  initialize;
  repeat
    repeat
      compute ⊖ (invalid terms);
      compute ⊕ (valid terms);
    until (no new results evaluated);
    compute ⧫ (symmetrical compensating terms);
    compute ◀ (left-hand compensating terms);
    compute ▶ (right-hand compensating terms);
  until (no new results evaluated);
  compute consistent_dependency_combinations;
end;
```

The algorithm mainly consists of a nested loop. In the inner loop the evaluation rules for valid ($z_m = \oplus$) and invalid terms ($z_m = \ominus$) are applied until no new results for x_1, y_1, x_2, and y_2 can be generated. Then the rules of the compensating terms ($z_m = \blacklozenge, \blacktriangleleft$, or $\blacktriangleright$) are executed. If this leads to further results, the loop for the valid and invalid terms is executed again. After the nested loop the results have to be adjusted with respect to compensation. In the following the single steps are described in detail:

Initializing: The variables $x_{(a,a)}$ and $y_{(a,a)}$ are initialized by $\oplus$, because for each termination dependency the abort of both transactions is valid.

After the initialization we continue with the steps of computing single terms for the four different values of z_m. In these steps of the algorithm we have to consider the following *general observation rules*:

1. Due to the fact that x_1 represents $x_{(a,a)}$ and $x_{(c,a)}$ and according to the definition of the termination dependencies (cf. Table 3), there are only the three possible values $\ominus, \oplus, \blacktriangleleft$ for x_1. In contrast, y_1 stands for $y_{(a,a)}$ and $y_{(a,c)}$. Thus, there are the following three possible values $\ominus, \oplus, \blacktriangleright$ for y_1.
2. The value $\blacklozenge$ appears only within $exc_{\blacklozenge}$ in the case both transactions commit. Therefore, only $x_{(c,c)}$ and $y_{(c,c)}$ are candidates for the value $\blacklozenge$.

Compute Invalid Terms ($z_m = \ominus$): To guarantee that z_m is invalid, the following term

$$\underbrace{(x_1 \wedge y_1) \vee (x_2 \wedge y_2)}_{\ominus}$$

has to evaluated to the invalid value $\ominus$ according to the rules presented in the following tables[1] which is a reduced version of Table 4:

[1] The $\star$-symbol stands for "don't care".

x	$\wedge$	y
$\ominus$	$\wedge$	$\star$
$\star$	$\wedge$	$\ominus$
$\oplus$	$\wedge$	$\blacktriangleleft$
$\blacktriangleright$	$\wedge$	$\blacktriangleleft$
$\blacktriangleright$	$\wedge$	$\oplus$
$\ominus$		

x	$\vee$	y
$\ominus$	$\vee$	$\ominus$
$\ominus$		

From the previous two tables we can derive the following evaluation rules:

- If an element of a conjunction is $\blacklozenge$, then the other one has to be $\ominus$.
- If x_1 (x_2) is $\blacktriangleleft$, then y_1 (y_2) has to be $\ominus$.
- If x_1 (x_2) is $\blacktriangleright$, then y_1 (y_2) has to be $\ominus, \oplus$, or$\blacktriangleleft$.
- If x_1 (x_2) is $\oplus$, then y_1 (y_2) has to be $\ominus$ or $\blacktriangleleft$.
- If y_1 (y_2) is $\blacktriangleleft$, then x_1 (x_2) has to be $\ominus$, $\oplus$, or $\blacktriangleright$.
- If y_1 (y_2) is $\blacktriangleright$, then x_1 (x_2) has to be $\ominus$.
- If y_1 (y_2) is $\oplus$, then x_1 (x_2) has to be $\ominus$ or $\blacktriangleright$.
- Due to the general observation rules $x_1, x_2 \in \{\ominus, \oplus, \blacktriangleleft\}$ and $y_1, y_2 \in \{\ominus, \oplus, \blacktriangleright\}$, x_1 or y_1 and x_2 or y_2 have to be $\ominus$.

Compute Valid Terms ($z_m = \oplus$): To guarantee that z_m is valid, the term $(x_1 \wedge y_1) \vee (x_2 \wedge y_2)$ has to be evaluated to the valid value $\oplus$ according to the rules presented in the following tables (cf. Table 4):

x	$\wedge$	y
$\oplus$	$\wedge$	$\oplus$
$\oplus$		

x	$\vee$	y
$\oplus$	$\vee$	$\star$
$\star$	$\vee$	$\oplus$
$\oplus$		

From these tables we can derive the following evaluation rules:

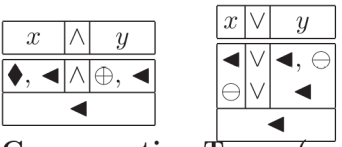
- If one conjunction is not $\oplus$, then the other one has to be $\oplus$. In this case, both elements of the latter conjunction has to be $\oplus$.
- If an element of each conjunction is $\oplus$, then one of the other has to be $\oplus$, too.

Compute Symmetrical Compensating Terms ($z_m = \blacklozenge$): To guarantee that z_m is compensatable in both directions, the term $(x_1 \wedge y_1) \vee (x_2 \wedge y_2)$ has to be evaluated to the compensating value $\blacklozenge$ using the rules presented in the following tables. From this rules we can directly derive a set of evaluation rules. Due to space restriction we refer to [8] for a detailed discussion.

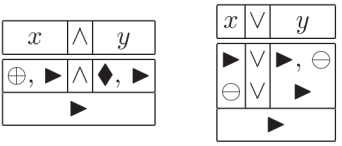
x	$\wedge$	y
$\blacklozenge, \blacktriangleleft$	$\wedge$	$\blacklozenge, \blacktriangleright$
$\blacklozenge$		

x	$\vee$	y
$\blacktriangleright$	$\vee$	$\blacklozenge, \blacktriangleleft$
$\blacktriangleleft$	$\vee$	$\blacklozenge, \blacktriangleright$
$\blacklozenge$	$\vee$	$\ominus, \blacklozenge, \blacktriangleleft, \blacktriangleright$
$\ominus$	$\vee$	$\blacklozenge$
$\blacklozenge$		

Compute Left-hand Compensating Terms ($z_m = \blacktriangleleft$): To guarantee that z_m is valid with a compensatable transaction t_i, the term $(x_1 \wedge y_1) \vee (x_2 \wedge y_2)$ has to be evaluated to the compensating value $\blacktriangleleft$ according to the rules in the following tables:



Compute Right-hand Compensating Terms ($z_m = \blacktriangleright$): To guarantee that z_m is valid with a compensatable transaction t_j, the term $(x_1 \wedge y_1) \vee (x_2 \wedge y_2)$ has to be evaluated to the right-hand compensatable value $\blacktriangleright$ according to the rules in the tables below and the related evaluation rules:



Compute Consistent Dependency Combinations: After the nested loop of the algorithm the results for the termination event combination of X and Y are combined to possible dependencies $X(t_i, t_k)$ and $Y(t_k, t_j)$. In this step, these set of dependencies are adjusted with respect to the compensation property. In case one of the involved transactions is compensatable, this property must be reflected in all corresponding dependencies. For example, if the transactions t_i and t_k are compensatable and vital-dependent for each other, then the related dependency is $vital\_dep_{\blacklozenge}(t_i, t_k)$ and not $vital\_dep_{\blacktriangleleft}(t_i, t_k)$, $vital\_dep_{\blacktriangleright}(t_i, t_k)$, or $vital\_dep(t_i, t_k)$. In conclusion, we can only build dependency combinations of $X(t_i, t_k)$ and $Y(t_k, t_j)$ where the compensation property of transaction t_k is reflected in both dependencies. For example, a combination of $vital\_dep_{\blacklozenge}(t_i, t_k)$ and $vital(t_k, t_j)$ is not valid. In contrast, $vital\_dep_{\blacktriangleleft}(t_i, t_k)$ and $vital(t_k, t_j)$ is a valid dependency combination. In this case t_k is non-compensatable.

The algorithm described in this subsection produces a set of rules including all reasonable and consistent dependency combinations (the full set of 120 rules can be found in [8]).

3.3 Sample Application of the Algorithm

The following example illustrates our algorithm for evaluating valid combinations of dependencies for a given transitive dependency. Let t_i, t_k, and t_j be transactions and $vital\_dep_{\blacklozenge}(t_i, t_j)$ the given dependency Z. According to Table 3, $z_{(a,a)}$ is set to $\oplus$, $z_{(a,c)}$ to $\blacktriangleright$, $z_{(c,a)}$ to $\blacktriangleleft$, and $z_{(c,c)}$ to $\oplus$. In this case we have to evaluate two valid, one right-hand and one left-hand compensating term.

1. **Initializing:** By definition the abort of both transactions is allowed for all dependencies, i.e., $x_{(a,a)}$ and $y_{(a,a)}$ are always initialized with $\oplus$. Thus, the following formulas has to be fulfilled:

$$(\underbrace{x_{(a,a)}}_{\oplus} \wedge \underbrace{y_{(a,a)}}_{\oplus}) \vee (x_{(a,c)} \wedge y_{(c,a)}) \Rightarrow \oplus$$

$$(\underbrace{x_{(a,a)}}_{\oplus} \wedge y_{(a,c)}) \vee (x_{(a,c)} \wedge y_{(c,c)}) \Rightarrow \blacktriangleright$$
$$(x_{(c,a)} \wedge \underbrace{y_{(a,a)}}_{\oplus}) \vee (x_{(c,c)} \wedge y_{(c,a)}) \Rightarrow \blacktriangleleft$$
$$(x_{(c,a)} \wedge y_{(a,c)}) \vee (x_{(c,c)} \wedge y_{(c,c)}) \Rightarrow \oplus$$

2. **Compute Valid Terms:** Because there are no invalid terms we start with the valid terms. The first valid term directly evaluates to $\oplus$ because the terms $(x_{(a,a)} \wedge y_{(a,a)})$ evaluates to $\oplus$ and the disjunction of a $\oplus$ and another value also evaluates to $\oplus$. Without any intermediate results for $x_{(c,a)}$, $y_{(a,c)}$, $x_{(c,c)}$, and $y_{(c,c)}$, we cannot derive whether the last formula is valid or not. Thus, we continue with the computation of the compensating terms.
3. **Compute Left-hand Compensating Term:** The term $(x_{(c,a)} \wedge y_{(a,a)})$ can only be evaluated to $\blacktriangleleft$, if $x_{(c,a)}$ is set to $\blacktriangleleft$ or $\blacklozenge$. However, the second general observation rule says that $x_{(c,c)}$ but not $x_{(c,a)}$ can be evaluated to $\blacklozenge$. Furthermore, $x_{(c,a)}$ may be set to $\ominus$ when the other term $(x_{(c,c)} \wedge y_{(c,a)})$ evaluates to $\blacktriangleleft$ (see the rules for left-hand compensating terms):

$$(\underbrace{x_{(c,a)}}_{\blacktriangleleft,\ominus} \wedge \underbrace{y_{(a,a)}}_{\oplus}) \vee \underbrace{(x_{(c,c)} \wedge y_{(c,a)})}_{\blacktriangleleft,\ominus} \Rightarrow \blacktriangleleft$$

4. **Compute Right-hand Compensating Term:** The evaluation of the right-hand compensating term is similar to the left-hand compensating term. The intermediate results are as follows:

$$(\underbrace{x_{(a,a)}}_{\oplus} \wedge \underbrace{y_{(a,c)}}_{\blacktriangleright,\ominus}) \vee \underbrace{(x_{(a,c)} \wedge y_{(c,c)})}_{\blacktriangleright,\ominus} \Rightarrow \blacktriangleright$$

   Next, we leave this loop and return to the first loop where the valid and invalid terms are evaluated.
5. **Compute Valid Term:** The disjunction of the last formula evaluates only to $\oplus$, if $(x_{(c,a)} \wedge y_{(a,c)})$ or $(x_{(c,c)} \wedge y_{(c,c)})$ evaluates to $\oplus$. However, two steps before we derived that $x_{(c,a)}$ may only be either $\blacktriangleleft$ or $\ominus$ and $y_{(a,c)}$ either $\blacktriangleright$ or $\ominus$. Thus, the conjunction $(x_{(c,c)} \wedge y_{(c,c)})$ has to be $\oplus$. This is done by setting both elements to $\oplus$:

$$(\underbrace{x_{(c,a)}}_{\blacktriangleleft,\ominus} \wedge \underbrace{y_{(a,c)}}_{\blacktriangleright,\ominus}) \vee \underbrace{\underbrace{(x_{(c,c)}}_{\oplus} \wedge \underbrace{y_{(c,c)})}_{\oplus}}_{\oplus} \Rightarrow \oplus$$

6. **Compute Left-hand Compensating Term:** The conjunction $(x_{(c,c)} \wedge y_{(c,a)})$ cannot be evaluated to $\blacktriangleleft$ because the first element $x_{(c,c)}$ is $\oplus$. Thus, this term has to evaluate to $\ominus$ and the other term $(x_{(c,a)} \wedge y_{(a,a)})$ to $\blacktriangleleft$. For that, $x_{(c,a)}$ is set to $\blacktriangleleft$. Furthermore, $y_{(c,a)}$ is set to $\ominus$ or $\blacktriangleleft$ to evaluate the

conjunction to $\ominus$:

$$\underbrace{\underbrace{(x_{(c,a)}}_{\blacktriangleleft} \wedge \underbrace{y_{(a,a)})}_{\oplus}}_{\blacktriangleleft} \vee \underbrace{\underbrace{(x_{(c,c)}}_{\oplus} \wedge \underbrace{y_{(c,a)})}_{\blacktriangleleft,\ominus}}_{\ominus} \Rightarrow \blacktriangleleft$$

7. **Compute Right-hand Compensating Term:** The evaluation of the right-hand compensating term is similar to the left-hand compensating term:

$$\underbrace{\underbrace{(x_{(a,a)}}_{\oplus} \wedge \underbrace{y_{(a,c)})}_{\blacktriangleright}}_{\blacktriangleright} \vee \underbrace{\underbrace{(x_{(a,c)}}_{\blacktriangleright,\ominus} \wedge \underbrace{y_{(c,c)})}_{\oplus}}_{\ominus} \Rightarrow \blacktriangleright$$

   Thus, there are two possibilities for the dependencies X and Y. The dependency X may be either $vital\_dep_{\blacktriangleleft}(t_i, t_k)$ or $vital\_dep_{\blacklozenge}(t_i, t_k)$ and the dependency Y may be either $vital\_dep_{\blacktriangleright}(t_k, t_j)$ or $vital\_dep_{\blacklozenge}(t_k, t_j)$. Thus, there are four possible dependency combinations for Z.
8. **Compute Consistent Dependency Combinations:** The combinations $vital\_dep_{\blacklozenge}(t_i, t_k)$ with $vital\_dep_{\blacktriangleright}(t_k, t_j)$ and $vital\_dep_{\blacktriangleleft}(t_i, t_k)$ with $vital\_dep_{\blacklozenge}(t_k, t_j)$ are invalid because transaction t_k is contradictory specified with respect to the compensation property. In one dependency t_k is defined as compensatable whereas in the other as non-compensatable. In conclusion, the following two valid dependency combinations hold:

$$vital\_dep_{\blacklozenge}(t_i, t_k) \wedge vital\_dep_{\blacklozenge}(t_k, t_j) \Rightarrow vital\_dep_{\blacklozenge}(t_i, t_j)$$
$$vital\_dep_{\blacktriangleleft}(t_i, t_k) \wedge vital\_dep_{\blacktriangleright}(t_k, t_j) \Rightarrow vital\_dep_{\blacklozenge}(t_i, t_j)$$

4 Sample Application of Transitive Dependencies

The following examples are intended to clarify the derivation of transitive termination dependencies in the transaction closure framework and the detection of contradictory dependency specifications. The transaction closure in our example can be considered as a workflow with special dependencies among the related transactions.

A commonly used example is a travel planning activity. In our example this activity is modeled as transaction closure with the transactions t_2, t_3, t_4, t_5, t_6, and the coordinating root transaction t_1. Transaction t_2 represents the flight reservation which is essential for the trip. Moreover, a room in the beach hotel (t_3) may be reserved including a car (t_5) and a diving course (t_6). If the diving course cannot be booked we also cannot book the beach hotel. An alternative to the beach hotel is the city hotel (t_4).

One possibility to model this scenario is graphically illustrated in Figure 2 where the arrows denote the direction of the abort dependencies. For example, $t_1 \longrightarrow t_3$ means that the abort of transaction t_1 leads to the abort of t_3. Thus, $vital(t_i, t_j)$ corresponds to $t_i \longrightarrow t_j$, $dep(t_i, t_j)$ to $t_i \longleftarrow t_j$, $vital\_dep(t_i, t_j)$

to $t_i \longleftrightarrow t_j$, and $indep(t_i, t_j)$ to $t_i \text{ — } t_j$. Since for an exclusive dependency a commit of one transactions leads to an abort of the other one, $exc(t_i, t_j)$ corresponds to $t_i \longleftrightarrow t_j$.

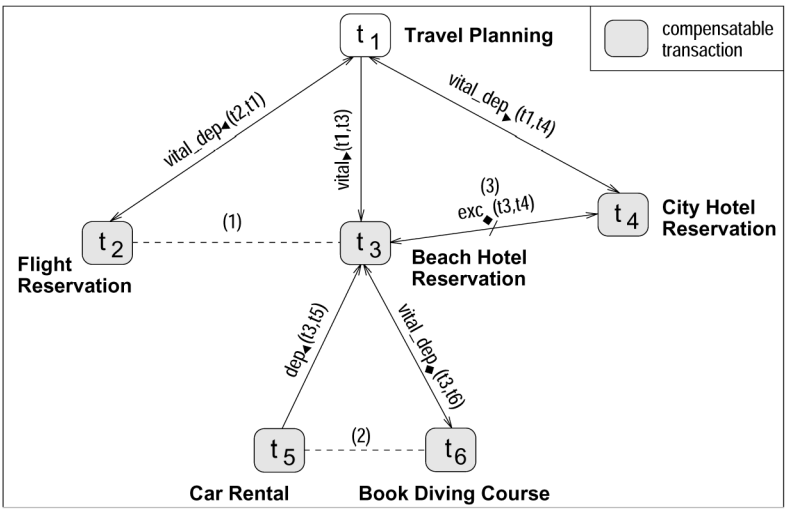


Fig. 2. A Sample Transaction Closure

For deriving the transitive and valid termination dependencies in the underlying transaction closure we use rules derived by the algorithm introduced in Section 3 and summarized in [8]. In the following, we discuss some interesting cases which refer to the dashed lines in Figure 2:

(1) We start with the consideration of the transactions t_1, t_2, and t_3. We know that t_1 is $vital\_dep_{\blacktriangleleft}$ on t_2 and $vital_{\blacktriangleright}$ for t_3. In this case an abort of transaction t_1 leads to an abort of t_2 which again result in an abort of t_3. However, the abort of t_3 has no influence on t_2 and thus on t_3. We can derive that transaction t_2 is *transitively* $vital_{\blacklozenge}$ for t_3:

$$vital\_dep_{\blacktriangleleft}(t_2, t_1) \wedge vital_{\blacktriangleright}(t_1, t_3) \Longrightarrow vital_{\blacklozenge}(t_2, t_3)$$

Hence, we may book the flight in case there is no free room in the beach hotel.

(2) A room in the beach hotel is only reserved (t_3) in case we can rent a car (t_5). The diving course (t_6) is also essential for the beach hotel reservation and vice versa. The dependency between the transactions t_5 and t_6 is evaluated by the following rule [8]:

$$vital\_dep_{\blacklozenge}(t_6, t_3) \wedge dep_{\blacktriangleleft}(t_3, t_5) \Longrightarrow dep_{\blacktriangleleft}(t_6, t_5)$$

Transaction t_6 is abort dependent of t_5 and, thus, the diving course can only be booked in case a car is rent.

(3) Another interesting dependency is the exclusive dependency between the two hotel reserving transactions t_3 and t_4. Moreover, we specified a vital dependency between t_1 and t_3 and t_1 is vital-dependent for t_4. We have to derive the transitive relationships between these three transactions to check whether the dependency specifications are compatible. The transitive dependency between the transactions t_4 and t_1 can be derived as follows:

$$vital_{\blacktriangleright}(t_1, t_3) \wedge exc_{\blacklozenge}(t_3, t_4) \Longrightarrow indep(t_1, t_4) \vee vital_{\blacktriangleright}(t_1, t_4) \vee exc_{\blacktriangleright}(t_1, t_4)$$

Here, the dependency can be specified by the transaction designer as either $indep(t_1, t_4)$, $vital_{\blacktriangleright}(t_1, t_4)$, or $exc_{\blacktriangleright}(t_1, t_4)$. However, we specified the $vital\_dep_{\blacktriangleright}(t_1, t_4)$. Thus, at least one of the specified dependencies between the transactions t_1, t_3, and t_4 has to be changed.
In our application scenario we identify the exclusive dependency between the transaction t_3 and t_4 as correct. In case the whole closure abort (e.g. we cannot book a flight), then the booking of a room in one of the hotels has to be aborted, too. This leads to the dependencies $vital_{\blacktriangleright}(t_1, t_3)$ and $vital_{\blacktriangleright}(t_1, t_4)$. The latter dependency is also compatible with the transitive dependency derived above.

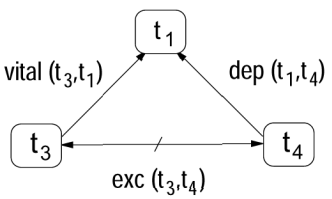


Fig. 3. Example of a Superfluous Transaction t_1

Another example for a contradictory specification is depicted in Figure 3. The dependencies between the transactions t_1, t_3, and t_4 in our application scenario could be specified as follows:

$$vital(t_3, t_1) \wedge exc(t_3, t_4) \wedge dep(t_1, t_4)$$

Dependency $exc(t_3, t_4)$ requires the abort of transaction t_4 in case transaction t_3 commits. Due to $dep(t_1, t_4)$, the abort of transaction t_4 leads to the abort of t_1. In the opposite case, if transaction t_4 commits, then t_3 and t_1 have to abort. The case that the transactions t_3 and t_4 abort leads also to the abort of t_1:

$$\begin{aligned} commit_{t_3} &\overset{exc(t_3,t_4)}{\Rightarrow} abort_{t_4} \overset{dep(t_1,t_4)}{\Rightarrow} abort_{t_1} \\ commit_{t_4} &\overset{exc(t_3,t_4)}{\Rightarrow} abort_{t_3} \overset{vital(t_3,t_1)}{\Rightarrow} abort_{t_1} \\ (abort_{t_3} \wedge abort_{t_4}) &\overset{vital(t_3,t_1)}{\Rightarrow} abort_{t_1} \end{aligned}$$

Due to the exclusive dependency, the commit of the transactions t_3 and t_4 is not allowed. As a consequence, transaction t_1 is aborted in any case. There is no rule which allows to specify such a dependency combination. Thus, we are able to detect such contradictory specification.

5 Conclusions and Outlook

In this paper, we have presented an algorithm to derive transitive termination dependencies under consideration of transaction compensation. Using the transitivity rules we are able to detect contradictory dependency specifications at design time. Furthermore, the transitivity rules can be used to analyze the effects of certain transaction termination events on other transactions. For instance, we can reason about a set of transactions that will be aborted in case a certain transaction aborts. The transitivity rules may also help the designer to find the exact relationship between two arbitrary transactions.

Our framework builds the basis for transaction design tools which can help in designing less failure-prone and more efficient applications. Currently, we are implementing a tool, called SpeCTraC[2], for designing and analyzing transaction closures. This tool will support the whole range of termination, execution, and object visibility dependencies of the transaction closure framework.

References

1. P. C. Attie, M. P. Singh, E. A. Emerson, A. Sheth, M. Rusinkiewicz. Scheduling Workflows by Enforcing Intertask Dependencies. *Distributed Systems Engineering*, 3(4):222–238, 1996. 114
2. A. Buchmann, M. T. Özsu, M. Hornick, D. Georgakopoulos, F. Manola. A Transaction Model for Active Distributed Object Systems. In A. K. Elmagarmid, ed., *Database Transaction Models for Advanced Applications*, pp. 123–151, Morgan Kaufmann, 1992. 113
3. P. K. Chrysanthis, K. Ramamritham. Synthesis of Extended Transaction Models Using ACTA. *ACM Transaction on Database Systems*, 19(3):450–491, 1994. 114
4. U. Dayal, M. Hsu, R. Ladin. A Transaction Model for Long-Running Activities. In G. M. Lohmann, A. Sernadas, and R. Camps, eds., *Proc. of the 17th Int. Conf. on Very Large Data Bases, VLDB'91*, pp. 113–122. Morgan Kaufmann, 1991. 113
5. H. Garcia-Molina. Using Semantic Knowledge for Transaction Processing in a Distributed Database. *ACM TODS*, 8(2):186–213, 1983. 115
6. D. Georgakopoulos, M. Hornick, P. Krychniak. An Environment for the Specification and Management of Extended Transactions in DOMS. In H.-J. Schek, A. P. Sheth, and B. D. Czejdo, eds., *RIDE-IMS'93, Proc. of the 3rd Int. Workshop on Research Issues in Data Engineering: Interoperability in Multidatabase Systems*, pp. 253–257. IEEE CS, 1993. 114
7. R. Günthör. The Dependency Manager: A Base Service for Transactional Workflow Management. In *RIDE'96, Proc. of the 6th Int. Workshop on Research Issues in Data Engineering: Interoperability in Nontraditional Database Systems*, pp. 86–95. IEEE CS, 1996. 114

[2] SPEcifying Consistent TRAnsaction Closures

8. K. Schwarz, C. Türker, G. Saake. Derived Transaction Termination Dependencies: An Algorithm for Computing Transitivity Rules. Preprint 7, Fakultät für Informatik, Universität Magdeburg, 1998. 120, 121, 124, 124
9. K. Schwarz, C. Türker, G. Saake. Transitive Dependencies in Transaction Closures. In B. Eaglestone, B. C. Desai, J. Shao, eds., *Proc. of the 1998 Int. Database Engineering and Applications Symposium (IDEAS'98)*, pp. 34–43. IEEE CS, 1998. 113, 114
10. M. P. Singh. Synthesizing Distributed Constrained Events from Transactional Workflow. In S. Y. W. Su, ed., *Proc. of the 12th IEEE Int. Conf. on Data Engineering, ICDE'96*, pp. 616–623. IEEE CS, 1996. 114
11. D. Worah, A. Sheth. Transactions in Transactional Workflows. In S. Jajodia, L. Kerschberg, eds., *Advanced Transaction Models and Architectures*, chapter 1, pp. 3–34, Kluwer, 1997. 113

A New Class Hierarchy Concurrency Control Technique in Object-Oriented Database Systems

Woochun Jun

Dept. of Computer Education
Seoul Natonal University of Education
Seoul, Korea
wocjun@ns.seoul-e.ac.kr

Abstract. Object-oriented databases (OODBs) have been used for many advanced applications requiring advanced modeling power. In this paper, a locking-based concurrency control scheme is presented for OODBs. The proposed scheme reduces locking overhead for class hierarchy which is an important property in OODBs. While existing schemes show good performance only for specific applications, the proposed scheme can incur less locking overhead for any applications. For the performance evaluation of the proposed scheme, simulation is conducted using 007 benchmark. Through the simulation, the proposed scheme is compared with the two existing techniques. The performance studies show that the proposed scheme is superior to existing works.

1 Introduction

Object-oriented databases (OODBs) have been adopted for managing non-standard applications such as computer-aided design (CAD) and computer-aided software engineering (CASE). These new applications require advanced modeling power in order to handle complex data and complex relationships among data. In an OODB, a class object consists of a group of instance objects and class definition objects. Also, there are two types of access to an object: instance access (instance read and instance write) and class definition access (class definition read and class definition write) ([2],[8]). An instance access consists of consultations and/or modifications of attribute values in an instance or a set of instances. A class definition access consists of consultations of class definition and/or modifications of class definition such as adding/deleting an attribute or a method code modification. Users can access objects by invoking methods.

In a database system, concurrency control regulates synchronization of access to the database so that it maintains the consistency of the database ([4],[22]). Usually, concurrency control enforces database consistency by satisfying an application-dependent correctness criterion among concurrent accesses on the same data item. *Serializability* is a widely used correctness criterion. Transactions are *serializable* if the interleaved execution of their operations produces the same output and has the same effects on the database as some serial execution of the same transactions ([3],

Eder et al. (Eds.): ADBIS´99, LNCS 1691, pp.128 -140, 1999.
© Springer-Verlag Berlin Heidelberg 1999

[4]). The locking-based scheme is the most widely used due to simplicity. In locking-based scheme, in order to check conflicts, commutativity is a criterion widely used to determine whether a method can run concurrently with methods in progress on the same object. Two methods commute if their execution orders are not important. Two methods conflict with each other if they do not commute.

Based on class hierarchy (also called inheritance hierarchy), a subclass inherits definitions defined on its superclasses. Also, an instance of a subclass is a specialization of its superclasses [20]. Due to class hierarchy, some operations involve access on class hierarchy rather than single class access. There are two types of operations on a class hierarchy: instance access to all or some instances of a given class and its subclasses, and class definition write. The author calls MCA (Multiple Class Access) for these two operations, and SCA (Single Class Access) for an operation to only one class such as class definition read and instance access to a single class. Thus, for a locking based concurrency control scheme, when a MCA access is requested on some class, say C, it is necessary to get locks for all subclasses of C as well as C.

In this paper, the author presents a locking-based concurrency control scheme for class hierarchy in OODBs. This paper is organized as follows. In Section 2, the author discusses previous studies. In Section 3, the author presents a new class hierarchy concurrency control scheme. For the performance evaluation of the proposed scheme, simulation model is constructed in Section 4. Then, a performance study is conducted by means of using 007 benchmark. The paper concludes with plans for future research in Section 5.

2 Related Work

As discussed in Section 1, due to inheritance, a MCA access involves more than one class on a class hierarchy. There are two major existing approaches to perform locking on a class hierarchy: *explicit locking* ([8],[14],[21]) and *implicit locking* ([14],[19],[20]).

In *explicit locking*, for MCA access on a class, say C, a lock is set not only on the class C, but also on each subclass of C on the class hierarchy. On the other hand, for SCA access to a single class, a lock is set for only the class to be accessed (called *target class*). Thus, for a MCA, transactions accessing a class near the leaf level of a class hierarchy will require fewer locks than transactions accessing a class near the root of a class hierarchy. Another advantage of *explicit locking* is that it can treat single inheritance where a class can inherit the class definition from one superclass, and multiple inheritance where a class can inherit the class definition from more than one superclass, in the same way. However, this technique increases the number of locks required by transactions accessing a class at a higher level in the class hierarchy.

In *implicit locking*, setting a lock on a class C requires extra locks (so called intention locks where they indicate that some lock is held on a subclass of the class) on a path from C to its root as well as on C. Theses intention locks are set on all the ancestors of a class before the target class is locked ([10],[18]). For MCA access on a class C, unlike explicit locking, locks are not required for every subclass of C. In this

case, it is sufficient to put a lock only on the target class (in single inheritance) or locks on the target class and subclasses of the target class which have more than one superclass (in multiple inheritance). Thus, it can reduce lock overhead over explicit locking. But, *implicit locking* requires a higher locking cost when a target class is near the leaf level in the class hierarchy due to intention lock overhead.

3 Proposed Class Hierarchy Locking Scheme

3.1. The Proposed Scheme

The objective of this work is to develop a new class hierarchy locking scheme which can be used for any OODB applications with less locking overhead than both existing schemes, explicit locking and implicit locking. To achieve this, author assigns some classes in the class hierarchy as *special classes.* Informally, the author defines a *special class* (SC) as MCA accesses are performed frequently. A formal way to determine if a class is a SC or not is as follows.

Assume that we have information on the number of accesses to each class by different transactions in an OODB. For the proposed scheme, we need to know only two types of number of accesses to each class: SCA and MCA. With this number of access information for each class, we determine if the class is assigned as a SC or not as follows. Note that this SC assignment scheme is based on bottom-up approach so that it starts from each leaf class until all classes are checked.

step 1) If a class, say C, is a leaf, then it becomes a non-SC.
If a class C has not been tested for SC assignment and all subclasses of C have been already tested, then do the followings
for class C and all of the subclasses,
calculate the number of locks (N1) when the class is assigned as a SC
calculate the number of locks (N2) when the class is assigned as a non-SC

step 2) Assign C as a SC only if N1 < N2 .
// C can be a SC only if the number of locks can be reduced //

For example, consider a simple class hierarchy as in Fig 1.a and assume that we have number of access information on the hierarchy as follows. (C_1: MCA=100, SCA=200, C_2: MCA=120, SCA=130; C_3: MCA=150, SCA=150; C_4: MCA=100, SCA=300, C_5: MCA=200, SCA=200, C_6: MCA=500, SCA=100, C_7: MCA=100, SCA=50, C_8: MCA=100, SCA=100). Note that the numbers represent the numbers of accesses to the class by different transactions. For example, 100 MCA accesses are performed on class C_8 and 100 SCA accesses on C_8, by different transactions accessing this class hierarchy. Note that, for MCA accesses, the numbers represent only access initiated at a given class. Thus, we do not count MCA access numbers initiated at its superclasses. In the proposed SC assignment scheme, since C_7 and C_8 are leaf classes, they are not assigned as SCs. At the class C_6, if C_6 is assigned as a SC, the number of locks needed for class C_6, C_7 and C_8 are 600 (for C_6), 300 (for C_7)

and 400 (for C_8), respectively, resulting 1300 locks for the three classes. On the other hand, if C_6 is not assigned as a SC, then the total number of locks needed for classes C_6, C_7, and C_8 are 1450 locks, where 1100 locks are from C_3 (1000 locks for MCA and 100 locks for SCA), 150 locks are from C_4 (100 locks for MCA and 50 locks for SCA), and 200 locks are from C_5 (100 locks for MCA and 100 locks for SCA). In this case, the proposed scheme works as in explicit locking. Thus, class C_3 becomes a SC. Similarly, the remaining classes become non-SCs. Fig. 1.b shows the result of the SC assignment scheme based on access frequency information.

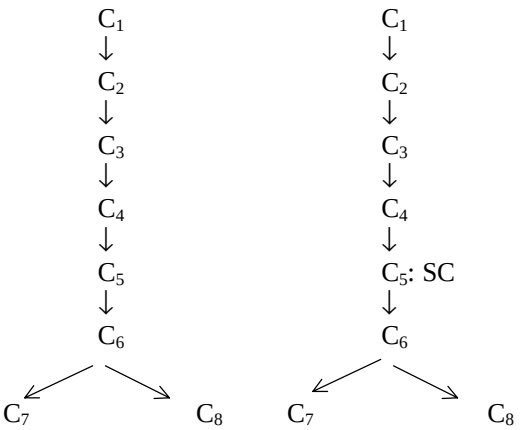


Fig.1.a. class hierarchy Fig.1.b. Results of SC assignment

The proposed locking-based concurrency control scheme is based on two-phase locking which requires each transaction to obtain a read (or write) lock on a data item before it reads (or writes) that data item, and not to obtain any more locks after it has released some lock [12]. For a given lock request on a class, say C, the author sets locks on C and all classes on the class hierarchy to which the class C belongs as follows (At this moment, single inheritance is assumed).

Step 1) Intention locking on SCs

- For each *SC* (if any) through the superclass chain of C, check conflicts and set an intention lock if it commutes. If it does not commute, block the lock requester.

Step 2) Locking on a target class

- If the lock request is a SCA access, check conflicts with locks by other transactions and set lock on only the target class C if it commutes and set an *instance-level* lock on the instance to be accessed if *a* method is invoked on the instance and commute. If it does not commute, block the requester.
- If the lock request is a MCA access, for each subclass of C, do as follows: from class C to the first *SC* (or leaf class if there is no SC) through the subclass chain of C, check conflicts and set a MCA lock on each class if commute. If the class C is a SC itself, then set a lock only on C.

Note that the proposed scheme sets a lock on the first SC so that other incoming transactions that access a subclass of the first SC can check conflicts since those transactions need to set intention locks on the first SC. Also, the reason to set a lock on each class from the class C to the first SC (excluding C and the first SC) is as follows. Any transactions accessing a class between C and the first SC (excluding C and the first SC) can get a lock regardless of the conflict with lock on C.

For simplicity, author adopts strict two-phase locking [22] which requires each transaction to release all the locks at the commitment time.

As an example, consider the following lock requests by two transactions T_1 and T_2 on a class hierarchy in Fig. 2.a.

a) T_1: MCA access on class C_5
b) T_2: SCA access on class C_3

Let L_i be a lock L set by transaction T_i. As seen in Fig 2.a 2.b, and 2.c, 3, 5 and 8 locks are required for T_1 and T_2 by the proposed scheme, explicit locking, and implicit locking, respectively.

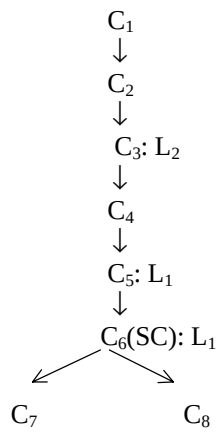


Fig. 2.a. Locks by the proposed scheme

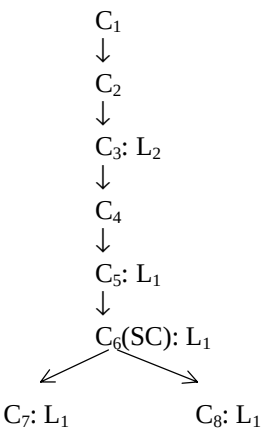


Fig. 2.b. Locks by explicit locking

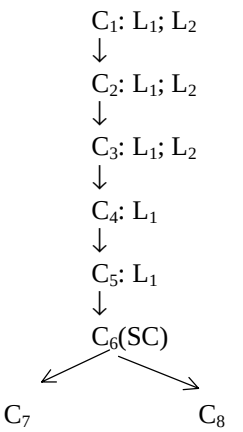


Fig. 2.c. Locks by implicit locking

3.2. The Proposed Scheme for Multiple Inheritance

For multiple inheritance, locks from the root to the target class are set by the rules as in single inheritance. But, for a MCA access in multiple inheritance, the proposed scheme differs from the implicit locking as follows. In implicit locking, locks are required for a target class and all subclasses of the target class which have more than one superclass. But, in the proposed scheme, locks are required for the target class and all subclasses of the target class which have more than one superclass and can be

accessed directly from classes other than class hierarchy rooted at the target class. Thus, in the proposed scheme, fewer locks are required than implicit locking.

For example, consider the class hierarchy in Fig. 3.a. Assume that a class definition needs to be changed in class F by transaction T_1. The implicit locking needs to get locks as in Fig. 3.a. On the other hand, locks are required as in Fig. 3.b if the proposed scheme is applied. In the proposed scheme, only classes H and I need to be locked since they can be reached directly from classes E and G, respectively, which are not classes in class hierarchy rooted at F.

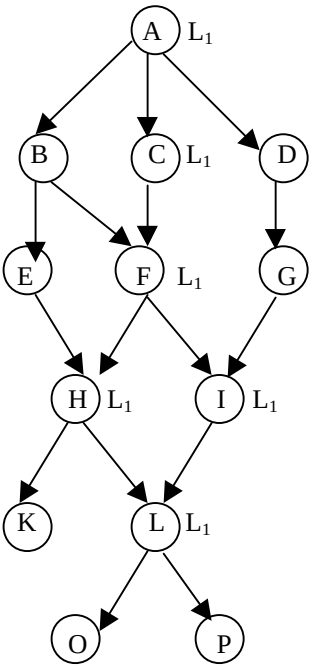


Fig. 3.a. Locks required by implicit locking

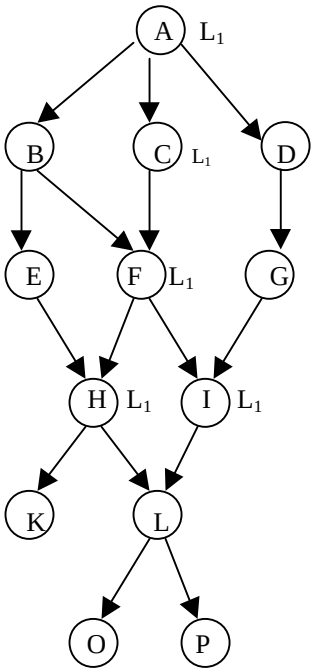


Fig. 3.b. Locks required by the proposed scheme

The author proves that the proposed algorithm is correct, that is, it satisfies serializability. The author proves this by showing that, for any lock requester, its conflict with a lock holder (if any) is always detected. With this proof, since the proposed scheme is based on two-phase locking, it is guaranteed that the proposed scheme satisfies serializability. The complete proof is shown in [16].

4 Performance Evaluation and Analysis by Simulation

In Section 4, performance evaluation and analysis are done by simulation. Especially, in Section 4.1, a simulation model is introduced. In Section 4.2, the simulation parameters and simulation methodology are discussed. In Section 4.3, simulation results from various testing cases and analysis are presented.

4.1. Simulation Model

The simulation model is constructed from models used in existing works for concurrency control performance evaluation [17]. Also, the simulation model is implemented using SLAM II simulation language [23]. Fig. 4 shows a general diagram of the simulation model. The simulation model has six major components: transaction generator, transaction manager, CPU scheduler, concurrency control manager (lock manager), deadlock manager and buffer manager. Also, it consists of two physical resources : CPU and memory.

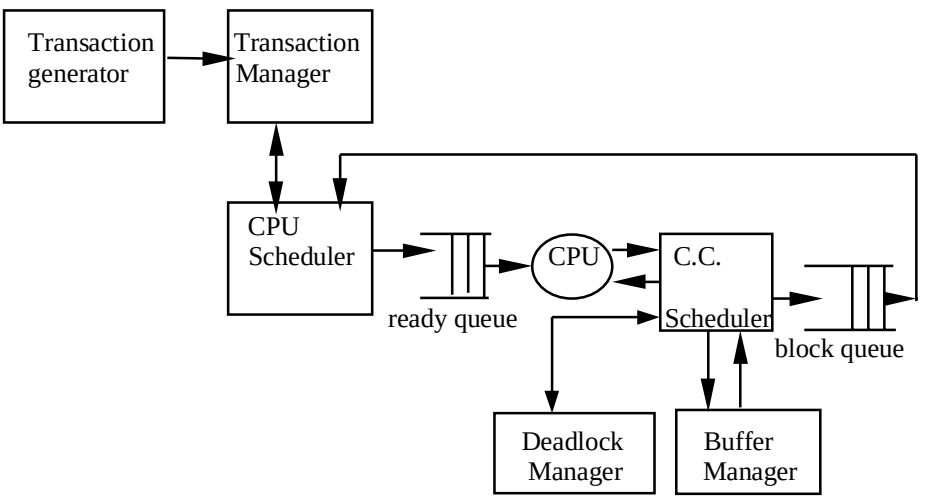


Fig. 4. Simulation Model [17]

The transaction generator creates each transaction with its creation time, unique transaction identifier and transaction type. Also, each transaction consists of a sequence of (method, object-id) pairs. The transaction manager is responsible for scheduling and executing all transactions. It sends lock/unlock requests as well as abort/commit messages to the concurrency control scheduler. It also restarts aborted transactions. The CPU scheduler performs various CPU-related operations such as executions of methods. The CPU can be released by a transaction as a result of a lock conflict or for an I/O operation. For simplicity, the FIFO (First-in First-out) is chosen

for CPU scheduling scheme. That is, any transaction arriving to CPU first has the higher priority. Also, any transaction holding CPU can not be preempted by other transactions. The concurrency control scheduler (CC scheduler) synchronizes data access requests among transactions. The CC scheduler orders the data accesses based on the concurrency control protocol executed. An access request of a transaction is either granted or results in blocking or abort of the transaction. If access request is granted, the transaction attempts to read the data item from the main memory (MM). The data access to MM is done through buffer manager. Since main-memory database is assumed in this simulation, there is no page fault. The FIFO strategy is used in the management of memory buffer. The deadlock manager (DM) detects any deadlock occurred during data item access. If a transaction is blocked for specific time period, DM is invoked to check a deadlock using WFG (wait-for graph). If a cycle is detected, then the transaction will be aborted and restarted.

4.2. Simulation Parameter and Methodology

The 007 benchmark is chosen in order to evaluate the proposed locking scheme in OODB There have been a number of benchmarks in OODB environments ([1],[5],[9]). But, existing benchmarks are not comprehensive so that wide range of OODB features can not be tested accordingly. For example, HyperModel [1] does not include object queries and repeated object updates. Also, it is difficult for testers to implement the model from their specifications.

The 007 benchmark ([6],[7]) provides a comprehensive test of OODB performance than its predecessors. Especially, it provides wide range of pointer traversal including sparse traversals and dense traversals, a rich set of updates and queries including sparse updates and the creation and deletion of objects. Also, its implementation details are open to public so that the benchmark can be implemented easily.

All the parameters used in simulation are summarized in Tables 1 and 2. Note that the OODB benchmark 007 is adopted to define database and transaction-related parameters. Also, all the parameters related to machine and disk are derived from the DEC 3000 Model 400/400S AXP Alpha workstation [11] and Micropolis 22000 disk drivers [24], respectively. The following notations are used to classify the simulation parameters accordingly.

M : Machine related parameters
D : Disk related parameters
TR: Transaction related parameters
DB: Database related parameters

Parameters	Default Value [Reference]
M: CPU power	140 MIPS [11]
M: time to process one operation	0.000007 ms [15]
M: mean time to set a lock by an instance access transaction	0.3641 ms (Implicit lock) [calculated] 0.3537 ms (Explicit lock) [calculated] 0.3572 ms (Proposed) [calculated]
M: mean time to release a lock by instance access transaction	0.0035 ms (Implicit lock) [calculated] 0.0019 ms (Explicit lock) [calculated] 0.0019 ms (Proposed) [calculated]
M: mean time to set a lock by class definition access transaction	0.3522 ms (Implicit lock) [calculated] 0.3522 ms (Explicit lock) [calculated] 0.3522 ms (Proposed) [calculated]
M: mean time to release a lock by class definition access transaction	0.0011 ms (Implicit lock) [calculated] 0.0011 ms (Explicit lock) [calculated] 0.0011 ms (Proposed) [calculated]
M: number of bytes per word	4 [11]
M: Memory word access time	0.00018 ms [11]
M: number of memory buffer	20 [11]
D: Size of disk(block) page	2048 bytes [24]
D: Avg. disk seek time	10 ms [24]
D: Avg. disk latency time	5.56 ms [24]
D: Disk page transfer time	0.0064 ms [24]
D: Number of pages in Database	1997 pages [Calculated]
DB: NumAtomicPerComp	20 [8]
DB: NumConnPerAtomic	3,6,9 [8]
DB: NumCompPerModule	500 [8]
DB: NumAssmPerAssm	3 [8]
DB: NumAssmLevels	7 [8]
DB: NumCompPerAssm	3 [8]
DB: NumModules	1 [8]
DB: Number of instances in Module	1[8]
DB: Number of instances in Manual	1 [8]
DB: Number of instances in Document	500 [8]
DB: Number of instances in Connection	30000 [8]
DB: Number of instances in AtomicPart	10000 [8]

Table 1. Static Parameters of the Simulation Model

Parameters	Default value (Range)
M: multiprogramming level	10 (5 - 15)
TR: Prob. of Traversal	0.45 (0 - 1)
TR: Prob. of Query	0.45 (0 - 1)
TR: Prob. of Structural Modification	0.1 (0 - 1)
TR: Prob. of Traversal T1 (Traversal type)	0.08 (0 - 1)
TR: Prob. of Traversal T6 (Traversal type)	0.08 (0 - 1)
TR: Prob. of Traversal T2 (Traversal type)	0.08 (0 - 1)
TR: Prob. of Traversal T3 (Traversal type)	0.08 (0 - 1)
TR: Prob. of Traversal T8 and T9 (Traversal type)	0.08 (0 - 1)
TR: Prob. of Traversal CU (Traversal type)	0.05 (0 - 1)
TR: Prob. of Query Q1 (Query type)	0.09 (0 - 1)
TR: Prob. of Query Q2, Q3 and Q7 (Query type)	0.09 (0 - 1)
TR: Prob. of Query Q4 (Query type)	0.09 (0 - 1)
TR: Prob. of Query Q5 (Query type)	0.09 (0 - 1)
TR: Prob. of Query Q8 (Query type)	0.09 (0 - 1)
TR: Prob. of Insert (Structural Modification)	0.05 (0 - 1)
TR: Prob. of Delete (Structural Modification)	0.05 (0 - 1)
TR: Transaction interarrival time	500 (100 – 1000)

Table 2. Dynamic Parameters of the Simulation Model

The 007 benchmark has three database sizes: small, medium and large. Each has different number of instance per class. For the simulation, small size is selected for simplicity. It is assumed that the transaction arrivals are based on the Poisson distribution. In the Poisson distribution, any transaction arrival time is totally random [13]. Also, in order to prevent system overload, the total number transactions in the system at any moment is limited by the parameter *Multiprogramming level*. As the performance metrics, average response time is adopted. The response time of a transaction is defined as follows.

Transaction response time = transaction commit time - transaction arrival time

4.3. Analysis

In order to evaluate the performance of the proposed concurrency control technique, two simulation testing cases are performed. For two testing cases, the proposed scheme and two existing concurrency control techniques, the explicit locking and the implicit locking, are tested. Also, analysis is done based on the simulation results.

Fig. 5 shows the testing case of varying arrival rate. The implicit locking performs the worst. It requires intention locks for superclasses of a target class, for any kind of

access. This results in lock overhead. The explicit locking does not require any intention locks. But, for MCA accesses, it may incur much overhead than implicit locking. On the other hand, in the proposed scheme, lock overhead is less than both implicit and explicit locking, using frequency information of each class. On the average, the proposed scheme works better than implicit locking by 52.6% and better than explicit by 32.9%. Also, explicit locking works better than the implicit locking by 26.5%.

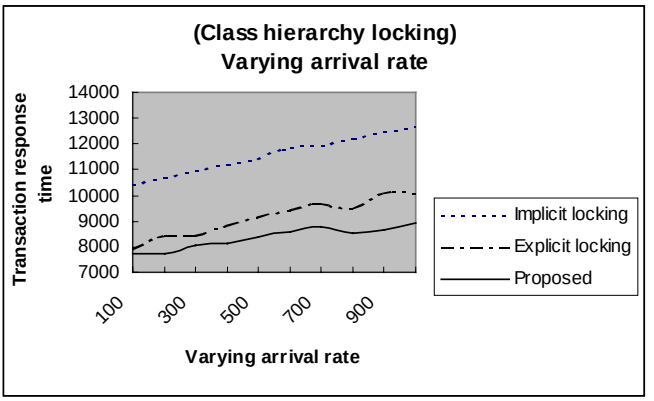


Fig. 5. Varying arrival rate

Fig. 6 shows the testing case of varying access to class hierarchy. In this testing case, transactions access from the root class to the leaf class in the class hierarchy. The purpose of this testing case is to measure the performance of class hierarchy locking technique used by each scheme as transactions access classes in the different levels of the class hierarchy.

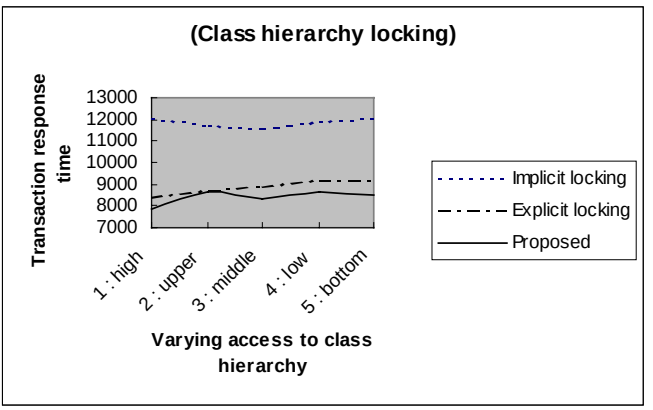


Fig. 6. Varying access to class hierarchy

As transactions access classes near the root in the class hierarchy, the implicit locking has less locking overhead while the explicit locking incurs much locking overhead for MCA accesses. On the other hand, if transactions access the leaf level in

the class hierarchy, the implicit locking incurs higher locking overhead due to intention locks while explicit locking takes small overhead. No matter where transactions access to class hierarchy, the proposed scheme performs better than both works. As shown in Fig. 6, there is not much difference as access to class hierarchy varies. This shows that locking overhead incurred by each scheme does not affect the performance significantly. On the average, the proposed scheme works better than the implicit locking by 40.91% and better than the explicit locking by 5.9%. The explicit locking works better than the implicit locking by 33%.

5 Conclusions and Further Work

In this paper, the author presents a concurrency control scheme for class hierarchy in OODBs. The proposed scheme adopts so called special class in order to reduce locking overhead. With assumption that number of accesses for each class is stable, the proposed scheme always incurs less number of locks than both explicit locking and implicit locking. In order to test the performance of the proposed technique, a simulation model was constructed and simulation experiments were conducted. Through the simulation experiments, it is concluded that the proposed scheme performs better than the implicit locking by 52.6 % and the explicit locking by 32.9%.

The proposed scheme aims at stable systems. But, if an OODB system whose schemas are continuously evolving, modifying special classes may incur overheads. Thus, the future research is to deal with evolving OODB systems. Also, this class hierarchy scheme needs to be combined with class composition hierarchy scheme for the complete concurrency control scheme. Currently, the author is developing a class composition hierarchy concurrency control scheme with special class concept.

References

1. Anderson, T., Berre, A., Mallison, M., Porter III, H., Scheider, B.: The HyperModel Benchmark, Proc. of the 2nd Int. Conf. on Extending Database Technology, Lecture Notes on Computer Science 416, Springer-Verlag, Berlin, (1990) 317 - 331
2. Agrawal, D. and Abbadi, A.: A Non-restrictive Concurrency Control for Object-Oriented Databases, 3rd Int. Conf. on Extending Data Base Technology, Vienna, Austria, Mar. (1992) 469 - 482
3. Bernstein, P. and Goodman, N.: Concurrency Control in Distributed Database Systems, *ACM Computing Surveys*, Vol. 13, No.2, (1981), 185 - 221
4. Berstein, P., Hadzilacos, V. and Goodman, N.: Concurrency Control and Recovery in Database Systems, Addison-Wesley, (1987)
5. Berre, A. and Anderson, T.: The HyperModel Benchmark for evaluating object oriented databases, In Object-Oriented Databases with Applications to CASE, Networks, and VLSI CAD, edited by R. Gupta and E. Horowitz, Englewood Cliffs, Jew Jersey, Prentice-Hall, (1991) 75 - 91

6. Carey, M. J., Dewitt, D. J. and Naughton, J. F: The 007 Benchmark, Proc. of the 1993 ACM SIGMOD Conference on Management of Data, Washington D.C., May, (1993) 12 - 21
7. Carey, M. J. Dewitt, D. J., Kant, C. and Naughton, F.: A Status Report on the 007 OODBMS Benchmarking Effort", Proc. of OOPSLA, Portland, Oregon, (1994) 414 - 426
8. Cart, M. and Ferrie, J.: Integrating concurrency control into an object-oriented database system, 2nd Int. Conf. on Extending Data Base Technology, Venice, Italy, Mar. (1990) 363 - 377
9. Cattell, R. and Skeen, J.: Object Operations Benchmark, ACM Transactions on Database Systems, Vol. 17, No. 1, Mar. (1992) 1 – 31
10. Date, C. J.: *An Introduction to Database Systems*, Vol. II, Addison-Wesley (1985)
11. DECdirect Workshop Solutions Catalog, winter 1993
12. Eswaran, K. P., Gray, J. N., Lorie, R. A. and Traiger, I. L.: The notion of consistency and predicate locks in a database system, Communication of ACM, Vol. 19, No.11 (1976) 624 - 633
13. Freund, J. E. and Walpole, R. E.: Mathematical Statistics, 4th edition, Prentice-Hall, Englewood Cliff, NJ, USA (1987)
14. Garza, J. F. and Kim, W.: Transaction Management in an Object-Oriented Database System, ACM SIGMOD Int. Conf. on Management of Data, Chicago, Illinois, Jun. (1988) 37 - 45
15. Huang, J.: Recovery Techniques in Real-Time Main Memory Databases, Ph.D. Dissertation, Univ. of Oklahoma, Dept. of Computer Science, Norman, Oklahoma, USA, (1995)
16. Jun W.: An Integrated Concurrency Control in Object-oriented Database Systems, Ph.D. Dissertation, Dept. of Computer Science, University of Oklahoma, Norman, Oklahoma, USA, (1997)
17. Kim, W., Chan, T. M. and Srivastava , J.: Processor Scheduling and Concurrency Control in Real-Time Main Memory Databases, IEEE Symposium on Applied Computing, Kansas City, Missouri, USA, Apr. (1991) 12 - 21
18. Korth, H. F. and Silberschartz, A.: Database System Concepts, 2nd Edition, McGraw Hill, Singapore (1991)
19. Lee, S. and Liou, R.: A Multi-Granularity Locking Model for Concurrency Control in Object-Oriented Database Systems, IEEE Trans. on Knowledge and Data Engineering, Vol. 8, No. 1, (1996) 144 - 156
20. Malta, C. and Martinez, J.: Limits of commutativity on abstract data types, *3rd* Int. Conf. on Information Systems and Management of Data, Bangalore, India, Jul. (1992) 261 - 270
21. Malta, C. and Martinez, J.: Automating Fine Concurrency Control in Object-Oriented Databases, 9th IEEE Conf. on Data Engineering, Vienna, Austria, Apr. (1993) 253- 260
22. Ozsu, M. T. and Valduriez, P.: Principles of Distributed Database Systems, Prentice Hall (1991)
23. Pritsker, A. A. B.: Introduction to Simulation and SLAM II, Systems Publishing Corporation. (1986)
24. 22000 Series - SCSI Micropolis Disk Drive Information

The G^r_Tree: The Use of Active Regions in G-Trees

Dimitris G. Kapopoulos and Michael Hatzopoulos

Department of Informatics, University of Athens,
Panepistimiopolis, Ilisia 157 84, Greece
{dkapo, mike}@di.uoa.gr

Abstract. In this paper we present an efficient data structure, called G^r_tree, for organizing multidimensional data. The proposed structure combines the features of distance functions of metric spaces and G-trees. Although the G^r_tree requires distance computations and has the overhead of a small amount of storage space, due to the introduction of active regions inside the partitions of the data space, it reduces the accesses of partial match and range queries. We give algorithms for the dynamic operations of the G^r_tree, examine several types of queries and provide some comparative results.

1 Introduction

B-trees are efficient data structures for indexing one-dimensional data. However, they are not suitable for indexing multidimensional data. Two well-known data structures that have been used for indexing multidimensional data are the K-D-B-tree [9] and the grid file [6]. In multidimensional data structures all attributes are treated in the same way and no distinction exists between primary and secondary keys. K-D-B-trees divide the *k*-dimensional data space into smaller *k*-dimensional regions that are organized into trees similar to B-trees. On the other hand, grid files divide a data space into a grid structure by splitting each dimension into several non-uniformly spaced intervals. The G-tree [5] combines the features of both B-trees and grid files. Similar structures have been proposed like the BD-tree [2], [3], the BANG file [4] and the zkdb tree [7], [8].

The *G-tree* is an efficient multidimensional and adaptable structure. It has the ability to adapt its shape to high dynamic data spaces and to non-uniformly distributed data. The G-tree divides the data space into a grid of variable size partitions. These partitions are stored in a B-tree-like organization. Only non-empty partitions, which span the data space, are stored.

The G-tree uses a variable-length partition numbering scheme. Each partition is assigned a unique *partition number* that is a binary string of 0's and 1's, where leading zeros are significant. Each partition corresponds to a disk block and data points are assigned to it until it is full. A full partition, *P* (parent), is split into two equal subpartitions *P*0 and *P*1 (children), e.g., if *P* = 101, then *P*0 = 1010 and *P*1 = 1011. The points of *P* are moved to *P*0 and *P*1. *P* is deleted from the G-tree, whereas *P*0 and *P*1,

Eder et al. (Eds.): ADBIS´99, LNCS 1691, pp.141 -155, 1999.
© Springer-Verlag Berlin Heidelberg 1999

if non-empty, are inserted into it. The new entry is assigned to the proper child P_c. If there is room, the new entry is added to P_c. Otherwise, P_c must be split. The splitting dimension alternates with a period equal to the number of dimensions, in a way that each dimension appears once in a cycle.

Fig. 1 shows the partitioning of a two-dimensional space with non-uniform distribution and data block capacity $BC = 2$. Its right side shows the equivalent binary representation of the partition numbers in decimal form. The conversion is made only for clarification. Internally, the partition numbers are stored as binary strings.

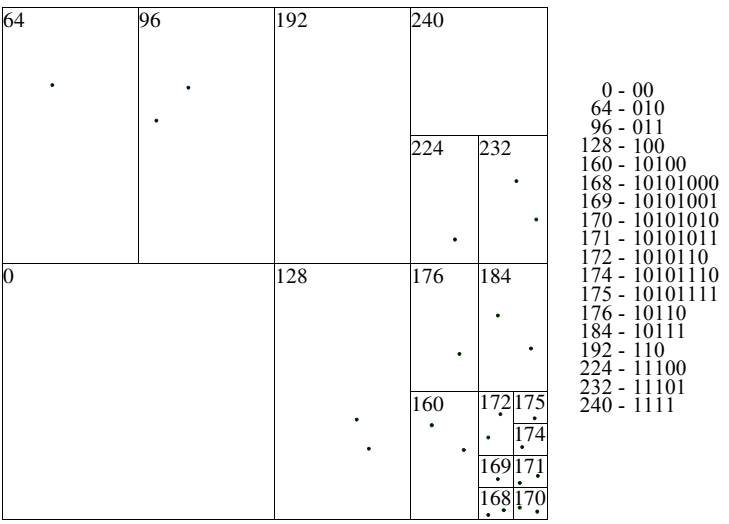


Fig. 1. The partition of a two-dimensional space

The G-tree arithmetic uses the operations $>$, $<$, $\subset$, $\subseteq$, $\supset$, $\supseteq$, *parent*(P) and *comp*(P). If the partitions P_1 and P_2 are b_1 and b_2 bits long, then b=min(b_1,b_2). $MSB(P_1,b)$ and $MSB(P_2,b)$ are the b most significant bits of P_1 and P_2 respectively.

- $P_1 > P_2$ if $MSB(P_1, b) > MSB(P_2, b)$. The partitions in the G-tree are non-overlapping and are totally ordered by the $>$ relation.
- $P_1 < P_2$ if $MSB(P_1, b) < MSB(P_2, b)$.
- $P_1 \subset P_2$ if $MSB(P_1, b) = MSB(P_2, b)$ and $b_1 > b_2$. That is, P_1 is a sub-partition of P_2.
- $P_1 \subseteq P_2$ if $P_1 \subset P_2$ or $P_1 = P_2$.
- $P_1 \supset P_2$ if $P_2 \subset P_1$. That is, P_1 is a super-partition of P_2.
- $P_1 \supseteq P_2$ if $P_2 \subseteq P_1$.
- The *parent*(P) is settled by removing the least significant bit from P.
- The *comp*(P) (complement) is settled by inverting the least significant bit of P.

A partition does not have to have a minimum number of points but when a data point of a partition P is deleted, the next step is to examine if the points of P and *comp*(P) can be stored in one data block. In such a case, the two partitions are merged into *parent*(P). P and *comp*(P) are removed from the G-tree, whilst *parent*(P) is inserted

into it. A leaf node entry in the G-tree consists of a partition number and a pointer to the block where the data points of this partition are stored. An inner node entry points to a block at the next lower level.

In [5], algorithms for the operations insert and delete are presented as well as for the processing of range queries. There is also a comparison with similar data structures such as the K-D-B-tree, the BD-tree, the zkdb-tree, the grid file and the multiattribute hashing. In this paper the advantages of the G-tree are examined.

We present the G^r_tree, a data structure that is a variation of the G-tree and aims to efficient organization of multidimensional data. The proposed structure combines the features of distance functions of metric spaces and G-trees. Despite the overhead of a small amount of storage space and some distance computations, the G^r_tree takes advantage of the use of active regions inside the partitions of the data space and reduces the number of accesses of partial match and range queries.

The organization of the paper is as follows: In Section 2, we introduce the G^r_tree. Then, in Section 3, we examine its dynamic behavior and give algorithms for insertion and deletion. In Section 4, we give searching algorithms and in Section 5 we present experimental results. We conclude this paper in Section 6 with a summary and some hints for future research.

2 The G^r_Tree

In this Section we present the G^r_tree and discuss such critical points as the active regions, the layers of the tree and its order.

2.1 The Active Regions

In order to organize a data space S through a G^r_tree, S must be considered as a *metric space M=(S,d)*, where d is a *distance function*. A distance function has the properties of symmetry, non-negativity and triangle inequality. That is, for every x, y in S there is a real number $d(x,y)$, called the distance of x from y, such that:

$$d(x,y) = d(y,x) \tag{1}$$

$$d(x,x) = 0 \ \wedge \ d(x,y) > 0, \quad \forall\, x \neq y. \tag{2}$$

$$d(x,y) \leq d(x,z) + d(z,y), \quad \forall\, z \in S. \tag{3}$$

Distance functions are used in access methods that aim to facilitate content-based retrieval applications [1], [10]. The k-dimensional space S^k of G^r_trees is a subset of the Euclidean space R^k, $k \geq 1$. We associate the *norm* of the difference of two points as the metric function for this space i.e.,

$$d(x,y) = |x - y|, \quad \forall x, y \in R^k \tag{4}$$

The norm is defined as:

$$|x| = \left(\sum_{j=1}^{k} x_j^2 \right)^{1/2}, \quad \forall x \in R^k \tag{5}$$

From (4) and (5) we have

$$|x - y| = \left(\sum_{j=1}^{k} (x_j - y_j)^2 \right)^{1/2}, \quad \forall x, y \in R^k \tag{6}$$

We assume that the data in each dimension are bounded i.e., for each point $x=(x_1,x_2,\ldots,x_k)$ we have that $l_{c_j} \leq x_j \leq h_{c_j}$, $1 \leq j \leq k$, where $l_c=(l_{c_1},l_{c_2},\ldots,l_{c_k})$ and $h_c=(h_{c_1},h_{c_2},\ldots,h_{c_k})$ are the leftmost and rightmost points of S^k.

The G^r_tree reduces the access cost of queries with the use of active regions inside the partitions. The active region of a partition P in S^k is defined by two areas that we note as $R(P)$ and $H(P)$. The first is a segment that seems like a part of a roll in the data space and is defined by an external and internal co-centric sphere. The second is a hyper-rectangle.

The *central point* $cp(P)=(cp_1(P),cp_2(P),\ldots,cp_k(P))$ of P is defined through the leftmost and rightmost corners of P. If these points are the $l_c(P)=(l_{c_1}(P),l_{c_2}(P),\ldots,l_{c_k}(P))$ and $h_c(P)=(h_{c_1}(P),h_{c_2}(P),\ldots,h_{c_k}(P))$ respectively, it is

$$cp_j(P) = \left(l_{c_j}(P) + h_{c_j}(P)\right)/2, \quad 1 \leq j \leq k \tag{7}$$

We define the *external sphere* $S_{ex}(P)$ of P, as the closed sphere $\overline{S}(cp(P), r_{ex}(P))$ with central point $cp(P)$ and radius $r_{ex}(P)$. The *external radius* $r_{ex}(P)$ is the maximum distance of any point of P from $cp(P)$. That is

$$S_{ex}(P) = \overline{S}(cp(P), r_{ex}(P)) = \left\{ x \in S^k : d(x, cp(P)) \leq r_{ex}(P) \right\} \tag{8}$$

The *internal sphere* $S_{in}(P)$ of P is defined as the open sphere $S(cp(P),r_{in}(P))$ with central point $cp(P)$ and radius $r_{in}(P)$. The *internal radius* $r_{in}(P)$ is the minimum distance of any point of P from $cp(P)$. That is

$$S_{in}(P) = S(cp(P), r_{in}(P)) = \left\{ x \in S^k : d(x, cp(P)) < r_{in}(P) \right\} \tag{9}$$

The *segment* $R(P)$ of P is defined as the complement of $S_{in}(P)$ to $S_{ex}(P)$. That is

$$R(P) = S_{ex}(P) - S_{in}(P) \tag{10}$$

We define the *minimum distance hyper-rectangle* $H(P)$ of P, as a k-dimensional hyper-rectangle that is enclosed in P and has its sides parallel to the corresponding sides of P. The distance of any side of $H(P)$ from the corresponding side of P is equal to the minimum distance of any point of P from any side of P. We note this minimum distance as $d_{min}(P)$. If there is a point on the sides of P, it is $d_{min}(P) = 0$.

The *active region* $AR(P)$ of a partition P is the intersection of the segment $R(P)$ and the minimum distance hyper-rectangle $H(P)$. That is

$$AR(P) = R(P) \cap H(P) \tag{11}$$

If $p \in P$ and $L_i(P)$ is the i-side of P, $1 \le i \le 2*k$, then it is

$$d(p, cp(P)) \le r_{ex}(P) \tag{12}$$

$$d(p, cp(P)) \ge r_{in}(P) \tag{13}$$

$$dist(p, L_i(P)) \ge d_{min}(P), \quad 1 \le i \le 2*k \tag{14}$$

The formulae of the distances $d(p,cp(P))$, $dist(p,L_i(P))$ and $d_{min}(P)$ are

$$d(p, cp(P)) = \left(\sum_{j=1}^{k} \left(p_j - cp_j(P) \right)^2 \right)^{1/2} \tag{15}$$

$$dist(p, L_i(P)) = \begin{cases} \left| p_i - l_{c_i}(P) \right|, & \text{if } 1 \le i \le k \\ \left| p_{i-k} - h_{c_{i-k}}(P) \right|, & \text{if } k < i \le 2*k \end{cases} \tag{16}$$

$$d_{min}(P) = \min_{p \in P} \left\{ \min_{1 \le j \le k} \left\{ \left| p_j - l_{c_j}(P) \right|, \left| p_j - h_{c_j}(P) \right| \right\} \right\} \tag{17}$$

Fig. 2 shows the active regions $AR(P_i)$ of the partitions P_i, i = 1, 2, of a data space in case of k=2. We use bold lines to show the sides of the partitions.

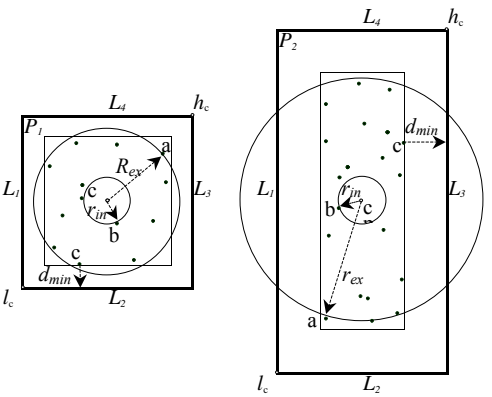


Fig. 2. The active regions $AR(P_1)$ *and* $AR(P_2)$

The distance of the point a and b from $cp(P_i)$ is the maximum and the minimum distance of any point of P_i from $cp(P_i)$. It is $r_{ex}(P_i)=d(cp(P_i),a)$ and $r_{in}(P_i)=d(cp(P_i),b)$. Moreover, the point c is the closest one to a partition side. The minimum distance of this point from a partition side is the value of the variable $d_{min}(P_i)$.

The active region $AR(P)$ is a subset of P that defines boundaries in the area of existing points. Its introduction aims to reduce the disk block accesses with the minimum overhead of storage space. We have chosen the schema of $AR(P)$ among many

alternatives. One of them was the hyper-rectangle defined by the external leftmost and rightmost points of *P*. As the external rightmost point of *P* we define the point whose the *j*-attribute, $1 \le j \le k$, has the maximum value of the values of the *j*-attribute of all data points. The definition for the external leftmost point is analogous. Both external leftmost and rightmost points are not necessarily data points. For the storage of this hyper-rectangle we need *2*k* numbers and the main drawback of this approach is that the space requirement depends on the number of dimensions.

The schema of *AR(P)* is dynamic because it depends on insertions, deletions and the distribution of points in *P*. The use of *AR(P)* is particularly useful when the data are not uniformly distributed in *P*. The G^r_tree may search for data only in *AR(P)* and this is its main difference from the G-tree. The overhead is the storage cost for the three real numbers $r_{ex}(P)$, $r_{in}(P)$ and $d_{min}(P)$ as well as some distance computations. However, as we will explain in the next Section, we do not store the above three numbers for all the partitions of the data space.

2.2 The G^r_Tree Layers

The G^r_tree consists of the *internal*, the *leaf* and the *active regions layer*. The first may have many levels and it is used for the access of the second layer where the partition numbers are stored in ascending order. In the leaf layer, we use links from right to left in order to facilitate sequential processing when we search for all the answers after the first match. In the third layer we store some of the active regions.

We use the notation *OA(P)* to declare the *overlap amount* of a partition *P* with its active region. The variable *OA(P)* is a real number in the range $0 < OA(P) \le 1$. The accurate calculation of *OA(P)* is difficult due to the schema of active regions. In our implementation we used the following formula that approximates *OA(P)* well.

$$OA(P) \approx \frac{\prod_{j=1}^{k} \min\left\{ r_{ex}(P) - r_{in}(P), \frac{DL_j(P)}{2} - d_{min}(P) \right\}}{\prod_{j=1}^{k} \frac{DL_j(P)}{2}} \qquad (18)$$

The notation $DL_j(P)$ declares the projection of *P* on the *j*-dimension. That is

$$DL_j(P) = \left| h_{c_j}(P) - l_{c_j}(P) \right|, \qquad 1 \le j \le k \qquad (19)$$

We store an *AR(P)* in the index if the *OA(P)* is lower than or equal to an *overlap factor ovf*. This factor is a parameter of the G^r_tree in the range $0 \le ovf < 1$. Lower value of *ovf* means less possibility to store an active region. On the other hand, low values of *ovf* for stored active regions, increase the possibility of avoiding access to their data blocks. In case of *ovf*=0, the layer of active regions does not exist.

Fig. 3 shows the three layers of the G^r_tree. The field *rec_num* in a leaf node declares the location (ascending order) of a partition in this node. This declaration is made only for clarification and it is not necessary to store this information. The field

ar_ex takes the value 0 or 1 and declares if the active region of a partition is stored in the third layer. An active region in a node of the third layer corresponds to the *rec_num*-partition of the *leaf_id*-leaf node. An active region may have entries from more than one subsequent leaf node. On the other hand, a leaf node may not have entries in more than one active region node. The capacity of leaf nodes may be different from the one of active region nodes. If an active region node has not the required space to store the entries from a leaf node, we use overflow blocks. This is very rare if the block sizes for leaf and active region nodes are the same. Moreover, the use of *ovf* restricts the number of the active regions that have to be stored.

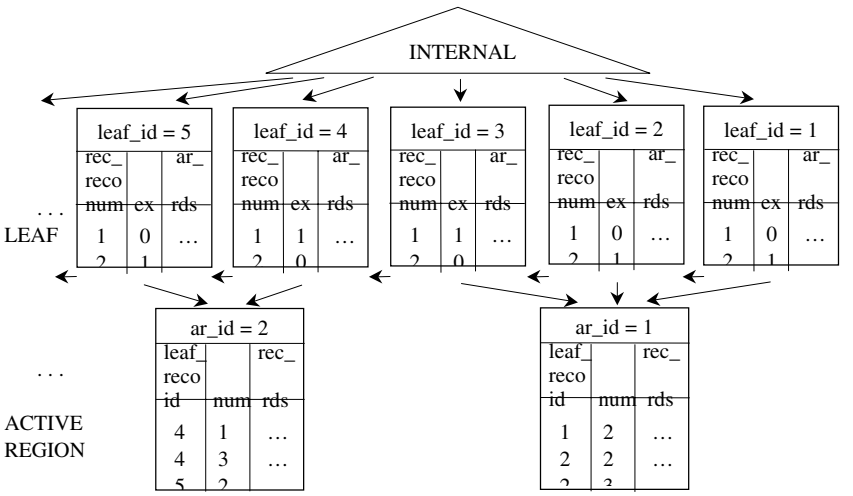


Fig. 3. The G^r_tree layers

The format of an internal node is (PR_0, P_1,PR_1, P_2,PR_2,…, P_n,PR_n). P_ι, $1 \le \iota \le n$, is the *i*-partition number and PR_ι, $0 \le \iota \le n$, is the address to a node in a lower level of the internal layer or to a leaf node. The format of a leaf node is (P_1,ar_ex_1,PR_1, P_2,ar_ex_2,PR_2,…, P_n,ar_ex_n,PR_n, *PRAR*). PR_ι, $1 \le \iota \le n$, is the *i*-addresses of the data block of the partition P_ι, and *PRAR* the address of the related active region node. Finally, the format of an active region node is ($leaf\_id_1$,rec_num_1,r_{ex_1},r_{in_1},d_{min_1}, $leaf\_id_2$,rec_num_2,r_{ex_2},r_{in_2},d_{min_2},…, $leaf\_id_n$,rec_num_n,r_{ex_n},r_{in_n},d_{min_n}, *PROV*), $1 \le n$, where *PROV* is the address of an overflow block in case of overflow.

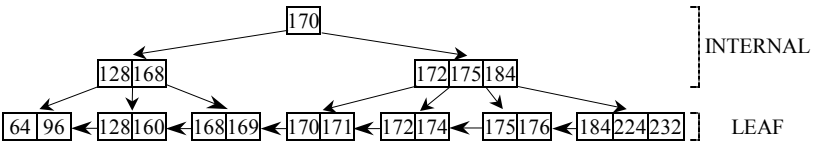


Fig. 4. The internal and leaf layer of the G^r_tree for the partitions of Fig. 1

Fig. 4 shows the internal and the leaf layer of the G^r_tree for the partitions of Fig. 1. As happens with the G-tree, the empty partitions 0, 192 and 240 are not stored.

2.3 The G^r_Tree Order

The number of entries in an internal and leaf node of the G^r_tree is not constant, as the length of partition numbers is variable. Moreover, the format of these nodes is different. Due to the above facts, we estimate the *average order m* of the G^r_tree. We define *m* through the average order of the internal layer m_{il} and the average order of the leaf layer m_{ll}. We use the following variables that are measured in bytes. *BS* is the block size and *PRS* is the size of pointers. *DS* is the size of the three distances r_{ex}, r_{in} and d_{min} that are real numbers. *PNS* is the average size of partition numbers. If the block size in internal and leaf layer is the same, we have

$$(m_{il} - 1) * PNS + m_{il} * PRS \leq BS \tag{20}$$

and

$$m_{il} = \left\lfloor \frac{BS + PNS}{PRS + PNS} \right\rfloor \tag{21}$$

As the storage of the variable *ar_ex* demands 1bit, it is

$$(m_{ll} - 1) * (PNS + 1/8) + (m_{ll} - 1) * PRS + PRS \leq BS \tag{22}$$

and

$$m_{ll} = \left\lfloor \frac{BS + PNS + 1/8}{PNS + PRS + 1/8} \right\rfloor \tag{23}$$

If *NIN* and *NLN* is the number of nodes in the internal and leaf layer respectively and *NN* is their sum, then we define *m* as

$$m = \frac{m_{il} * NIN + m_{ll} * NLN}{NN} \tag{24}$$

If *h* and d_{il} is the height of the tree and the average number of descendants of nodes in the internal layer respectively, it is

$$\left\lceil \frac{m_{il}}{2} \right\rceil \le d_{il} \le m_{il} \tag{25}$$

$$NIN = \sum_{j=0}^{h-2} d_{il}{}^{j} = \frac{d_{il}^{h-1} - 1}{d_{il} - 1} \tag{26}$$

$$NLN = d_{il}^{h-1} \tag{27}$$

$$NN = \frac{d_{il}^{h} - 1}{d_{il} - 1} \tag{28}$$

From (24), (26), (27) and (28) we have

$$m = \frac{m_{il} * \left(d_{il}^{h-1} - 1\right) + m_{ll} * d_{il}^{h-1} * \left(d_{il} - 1\right)}{d_{il}^{h} - 1} \tag{29}$$

3 Update Operations

Update operations of the G^r_tree have an additional cost in comparison with the G-tree. This is justified by the change that may cause the insertion or the deletion of a data point to the active region of its partition. In the following we describe the procedures for inserting and deleting multidimensional points into the G^r_tree.

3.1 Insertion

To insert a point p in the file, we first compute, as in the G-tree, its initial partition P_{init} and then the actual partition P_{act} where p should be inserted. The data block of P_{act} is transferred in main memory and if there is space we insert p and write the block back to disk. In the case that $AR(P_{act})$ has been stored, we compute the distance $d(cp(P_{act}),p)$ and the minimum distance of p from the sides of P_{act}. Then, we access the active region node and examine if $p \notin AR(P_{act})$. If it is true the inserted point alters the active region. If it is still $OA(P) \le ovf$, we update the active region node as at least one of $r_{ex}(P_{act})$, $r_{in}(P_{act})$ and $d_{min}(P_{act})$ has been changed. If it is $OA(P) > ovf$, we delete the $AR(P_{act})$ from its block and update the field *ar_ex* of the leaf node. If the data block is full we have a case of overflow and we split the block, as in the G-tree.

After a split the two children nodes share the active region node of their father. In case of overflow of an active region node, we create a new block and share active regions between the new and the old block. Then we update the leaf nodes the addresses of which reside in the field *leaf_id* of the new active region node. Alternatively, before we split and create a new active region block, we can check if there is space in the next or the previous block. If this is the case, then we can avoid the creation of a new block.

If the attributes of two points are integers or real numbers, then the calculation of their distance is obvious. For strings we use an algorithm that transforms them to unsigned long integers. This means that distance calculations are arithmetic for all

data types. There is a small possibility two different strings to be transformed to the same long integer. The possibility of an impossible partition split, due to the mapping of all data points to the same point, is low. This possibility is inversely proportional to the increase in the number of dimensions. We handle the case of an impossible split with overflow data blocks.

The G^r_tree may use a policy of *further splits* in order to organize better the data space. This policy tries to create the smallest partitions should all the points of a non-overflow partition belong to one of its future children. This aims to reduce as much as possible the overlap of partitions with the region of queries. An additional split is triggered if there is more than one data point in the partition. The policy of further splits has the drawback of creating longer partition numbers. This means additional storage space. Moreover, as our experimental results show, the number of partitions increases in case of a dynamic environment. This is due to the fact that points are inserted in areas that do not exist in the G^r_tree. The use of the above policy is parametric and is useful in a static or almost static environment. If this policy is not used and it is *ovf*=0, the G^r_tree becomes identical to the G-tree. Fig. 5 shows the policy of further splits for the partitions of Fig. 1. The dash lines show the additional splits. The partitions 96, 128 and 160 are replaced with the 112, 140 and 164 respectively. The empty partitions 0, 96, 116, 120, 128, 136, 142, 144, 160, 192 and 240 are not stored in the index.

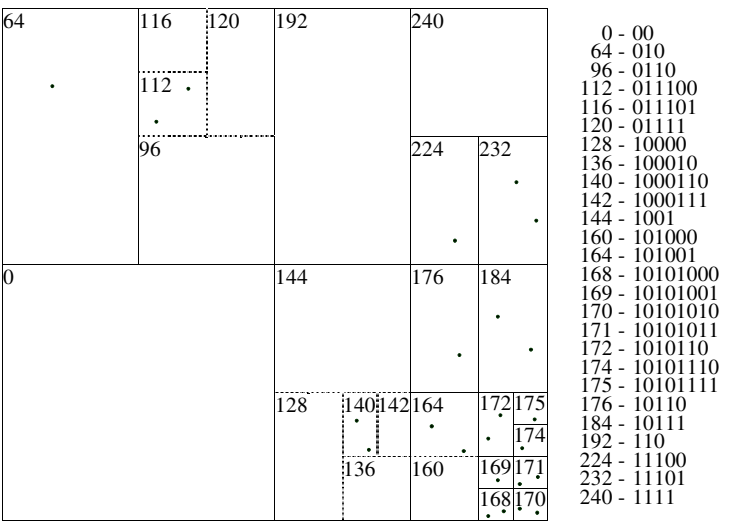


Fig. 5. Further splits for the partitions of Fig. 1

3.2 Deletion

For the deletion of a point p we first compute P_{init} and then examine if P_{act} exists in the index. If it is true, we examine if $p \in AR(P_{act})$. If no, the point does not exist and the

procedure stops. If yes, we delete the point and rewrite the data block to disk. We examine if the deletion of the point has altered the $AR(P_{act})$. If this is the case and before the deletion was $OA(P_{act}) \leq ovf$, we update the active region of P_{act} as it is already stored in the third layer. If it was $OA(P_{act}) > ovf$ but after the deletion is $OA(P_{act}) \leq ovf$, we insert the active region of P_{act} in the proper node.

In order to have blocks filled over a threshold, we try to merge a partition P with $comp(P)$ if after a deletion from P it is left with less than half points. In addition, we update the active regions, should they exist.

4 Search

In this section we examine how partial match and range queries are processed using the G^r_tree. For exact match queries we use the same technique as in the G-tree. In these queries we do not examine the active region node of a searched point p because this may cause one additional access when $p \in AR(P_{act})$. In contrast, the access of an active region node in partial match and range queries may save accesses to data nodes as an active region node includes the active regions of many partitions.

A *range query* is geometrically equivalent to a hyper-rectangle QR in the data space. We compute the partitions P_L and P_H that correspond to the leftmost and rightmost corners of QR. The length of the numbers of these partitions must be equal to the length of the number of the minimum size partition in the G^r_tree. We search for the partitions with numbers lower than or equal to the partition number of P_H and greater than or equal to the partition number of P_L. First, we search the index for P_H and then we continue the movement in the leaf layer from right to left. If P_L exists, we stop the search for possible qualified partitions. Otherwise, we continue until to find a super-partition (ascendant) of P_L with length greater than or equal to the length of the number of the maximum size partition (minimum binary string length) in the G^r_tree. We denote this descendant as $P_{L'}$. If we find a partition number lower than the one of $P_{L'}$ or we reach the first partition of the first leaf node and this is not $P_{L'}$, then $P_{L'}$ does not exist. As is concluded from the above, the minimum search interval is $[P_L, P_H]$ and the maximum $[P_{L'}, P_H]$. The extension of the search until $P_{L'}$ ensures that we find all the partitions that overlap with QR. This is different with the corresponding algorithm presented in [5]. In this algorithm the search is within $[P_L, P_H]$ and there is a possibility to lose qualified partitions in the case that P_L does not exist in the index.

Each one of the partitions resulting from the above search is examined for overlap with QR. If QR includes a partition, then all of its points are answers. If QR does not include the partition but there is an overlap between them, then the partition becomes a member of a list that contains possible qualified partitions. We visit an active region node if the number of the partitions in the above list that are related to this node is greater than or equal to an integer factor that we call *lower bound of overlaps*, and denote as *lbo*. If this is the case, we access the active region node and examine if the active region of the partition overlaps with QR. If there is no overlap, the partition is

removed from the list. We examine the data blocks of the remaining partitions for qualifying points.

The use of *lbo* increases the possibility of saving accesses to data blocks by first visiting an active region node. The value of *lbo* as well as the one of *ovf* may influence seriously the performance of the G^r_tree.

The procedure to answer a *partial match query* is similar to that of a range query, as the set of partial match queries is a subset of the range queries. If *i,* $1 \le i \le k$, is a dimension that participates in a partial match query with free value, then the upper and lower corners of *QR*, relative to this dimension, take the upper and lower value of their domain respectively.

5 Experimental Results

In this section we provide experimental results on the performance of the G^r_tree and compare them to the corresponding performance of the G-tree. We implemented the two structures in C on a SUN SPARCserver 1000 under SunOS 5.5 and in Borland C++ on a Pentium II under Windows 98. We used separate files for the indexes and the data. Each layer of the G^r_tree corresponded to one file, whereas for the G-tree we used one file for the internal nodes and one for the leaf nodes. Each file had its own block size. The number of attributes was four and all of them were included in the index. The type of attributes was long int. The block size *BS* was 1 Kbyte for all files. The data followed the normal distribution with mean value 0 and variation $5*10^6$. The range of values was $[10^5,10^6]$ and the step of increment 10^5. Similar experiments with other distributions showed that results depend very slightly upon the nature of the distribution from which the data is drawn. For the factors *ovf* and *lbo* of the G^r_tree we used the values *ovf*=0.9 and *lbo*=$\lceil BC_L/10 \rceil$, where BC_L is the average block capacity of leaf nodes.

The percentage average utilization of the internal and leaf nodes of both indexes for 10^6 records was approximately 69%, whereas their data node utilization for the same volume of data was approximately 65%. The utilization of the active region nodes was 94%. This high value of utilization is due to the fact that an active region node may have entries that correspond to many G^r_tree leaf nodes. Fig. 6 presents the space requirements of the two tree structures as a function of the number of records. As shown in this figure, the G^r_tree requires more space for its storage than the G-tree due to the existence of the active region layer. The amount of extra space depends on the factor *ovf* and it increases with the increment of *ovf*.

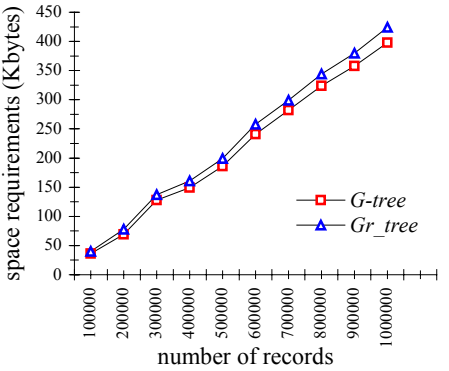


Fig. 6. Space requirements of indexes versus number of records

The following results correspond to the average disk accesses using 100 queries of the same type. The queries are taken uniformly from the insertion file. That is, the constant values of a partial match query over a file of *NR* records were taken from the places $\lfloor NR/100 \rfloor * j$, $1 \le j \le 100$, of the insertion file.

Fig. 7 shows the number of disk block accesses required for exact match queries versus the number of records. Both structures have the same performance. This was expected as both trees have the same height and the G[r]_tree does not use active regions for these queries.

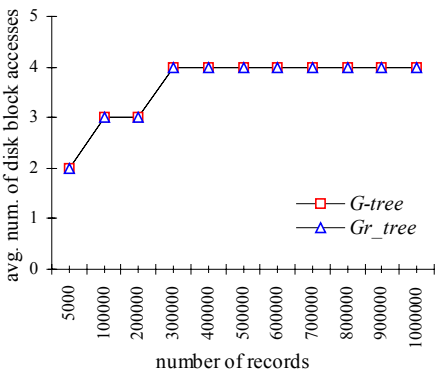


Fig. 7. Average number of disk block accesses per exact match query versus number of records

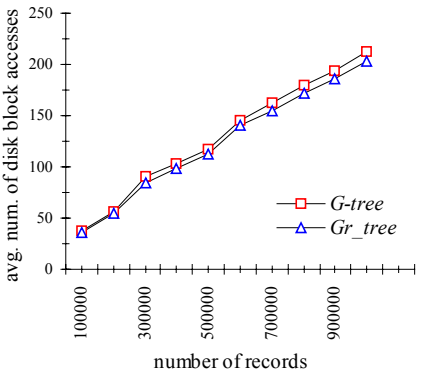


Fig. 8. Average number of disk block accesses per partial match query of one variable versus number of records

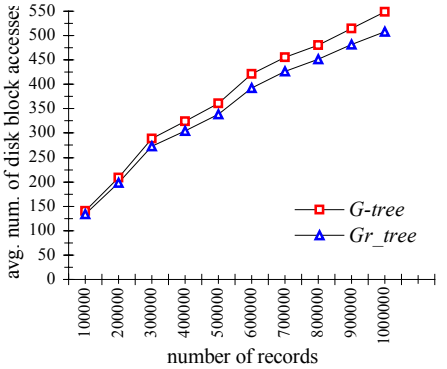


Fig. 9. Average number of disk block accesses per partial match query of two variables versus number of records

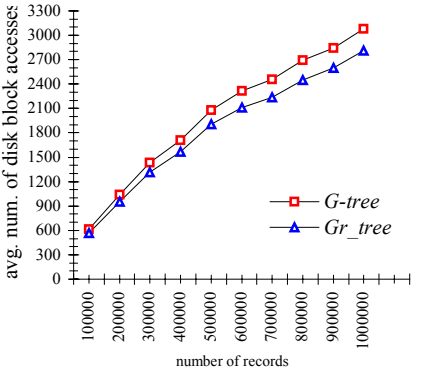


Fig. 10. Average number of disk block accesses per partial match query of three variables versus number of records

Fig. 8, Fig. 9 and Fig. 10 show the number of disk accesses required for partial match queries versus the number of records. These figures refer to queries with one, two and three variables respectively. The disk accesses required by the G^r_tree are fewer than the ones of the G-tree. The proposed structure takes more advantage of the G-tree with the increment of the number of variables and the number of records.

On the above grounds, we believe that the G^r_tree has better performance than the G-tree and, consequently, that the proposed structure efficiently extends the set of multidimensional data structures.

6 Summary

In this paper we presented the G^r_tree, a new balanced dynamic index structure for multidimensional data that combines the features of distance functions of metric spaces and G-trees. The design of the proposed structure is an attempt to minimize disk accesses in partial match and range queries. Despite of the overhead of a small amount of storage space and some distance computations, the G^r_tree takes advantage of the use of active regions inside the partitions of the data space and achieves its design motivation. We presented algorithms for inserting deleting and searching data from the G^r_tree and we reported the results of a comparison between the new structure and the G-tree.

Planned research work includes the optimization of the factors *ovf* and *lbo*, in order to improve the performance of the G^r_tree. Moreover, we believe that the use of active regions may be adapted by many other data structures and, consequently, improve their performance.

References

1. Ciaccia, P., Patella, M., Zezula, P.: M-tree: An Efficient Access Method for Similarity Search in Metric Spaces. Proc. 23th Athens Intern. Conf. on VLDB (1997) 426-435
2. Dandamundi, S., Sorenson, P.: An empirical performance comparison of some variations of the k-d tree and BD-tree. Int.J.Comput. Inform.Sci., vol.14 (1985) 135-159
3. Dandamundi, S., Sorenson, P.: Algorithms for BD-trees. Software Practice and Experience, vol.16 (1986) 1077-1096
4. Freeston, M.: The BANG file: a new kind of grid file. Proc. ACM, SIGMOD Conf. (1987) 260-269
5. Kumar, A.: G-Tree: A New Data Structure for Organizing Multidimentional Data. IEEE, Trans. On Knowledge and Data Eng., vol. 6, no. 2 (1994) 341-347
6. Nievergelt, J., Hintenberger, H., Sevcik, K.C.: The Grid File: an adaptable, symmetric multikey file structure. ACM Trans. Database Syst., vol. 9, no. 1 (1984) 38-71
7. Orenstein, J., Merrett, T.: A class of data structures for associative searching. Proc. ACM SIGACT-SIGMOD Symp. Principles of Database Systems (1984) 181-190
8. Orenstein, J.: Spatial query processing in an object-oriented database system. Proc. ACM SIGMOD Conf., Washington (1986) 326-336
9. Robinson, J.T.: The K-D-B-tree: A Search Structure for Large Multidimensional Dynamic Indexes. Proc. ACM SIGMOD Conf., Ann Arbor, MI (1981) 10-18
10. Samet, H.: Spatial Databases. Proc. 23th Athens Intern. Conf. on VLDB (1997) 63-129

S*–Tree: An Improved S+-Tree for Coloured Images*

Enrico Nardelli[1,2] and Guido Proietti[1]

[1] Dipartimento di Matematica Pura ed Applicata, Università di L'Aquila,
Via Vetoio, 67010 L'Aquila, Italy
{nardelli,proietti}@univaq.it
[2] Ist. di Analisi dei Sistemi e Informatica,
CNR, V.le Manzoni 30, 00185 Roma, Italy

Abstract. In this paper we propose and analyze a new spatial access method, namely the S*-tree, for the efficient secondary memory encoding and manipulation of images containing multiple non-overlapping features (i.e., coloured images). We show that the S*-tree is more space efficient than its precursor, namely the S+-tree, which was explicitly designed for binary images, and whose straightforward extension to coloured images can lead to large space wastage. Moreover, we tested time efficiency of the S*-tree in answering classical window queries, comparing it against a previous efficient access method, namely the HL-quadtree [7]. Our experiments show that the S*-tree can reach up to a 30% of time saving.

Keywords: Spatial databases, bintree, quadtree, window queries.

1 Introduction

In this work we focus on secondary memory representations of images containing multiple non-overlapping spatial features, like for instance agricultural maps, thematic maps, satellite views and many others. This is a very hot research topic, especially with the increasing interest of the database community towards the development of efficient management systems for geographical data (GIS). Therefore, we are implicitly assuming that the underlying images have all the peculiar aspects of images containing *region data*, and specifically the most prominent one, that is the *aggregation* of pixels of a given colour into patches. This induces a couple of observations: first, the number of features (i.e., colours) in the representing picture is limited (generally, from 8 to 32), second, and perhaps more important, it makes sense to apply hierarchical methods of representation of the image to save space and time.

One of the most successful hierarchical strategy for representing images containing region data is based on the decomposition of the image space containing the data into recursively nested subimages, until an homogeneous pattern is

* This research was partially supported by the CHOROCHRONOS TMR Program of the European Community.

Eder et al. (Eds.): ADBIS'99, LNCS 1691, pp. 156–168, 1999.
© Springer-Verlag Berlin Heidelberg 1999

obtained. The most popular decomposition techniques are the *binary decomposition* (which splits the image into two equal parts alternating an horizontal and a vertical subdivision) and the *quaternary decomposition* (which splits the image into four equal quadrants). The corresponding main memory representations of such split policies are the *bintree* [12] and the *region quadtree* [9]. Both data structures are easy to implement in main memory. On the other hand, when a secondary memory representation is needed (which is usually the case, given the large amount of data to be stored), things become more complicated. The problem is that of mapping a 2-dimensional set onto a 1-dimensional universe, while attempting to preserve as much as possible spatial proximity properties.

For images containing multiple non-overlapping features (for the sake of brevity, *coloured images* in the following, even though this term could be misleading, since it does not convey the concept that the underlying image is representative of region data and therefore well-suited to be managed by hierarchical spatial data structures), a number of different secondary memory implementations have been proposed. These can be subdivide into two categories: *leafcode representations*, obtained as a collection of the leaf nodes in the tree (such as, for example, the *linear quadtree* [5]), and *treecode representations*, obtained by a preorder tree traversal of the nodes in the quadtree (also called *DF-expressions* [6]). The latter approach is asymptotically more compact than the former one, but it has suffered for a long time the lacking of a paged version able to support the access to a given element without being forced to scan, in the worst case, the entire database. This difficulty have been overcome by de Jonge et al. [3], who developed the *S+-tree*, a spatial access method combining the advantages of treecode and leafcode representations, essentially by indexing through locational codes the space-compact DF-expression. However, as we shall see in the rest of the paper, the S+-tree is tailored on binary images, and a straightforward extension of it to coloured images has a severe space utilization drawback, which affects in its turn the time efficiency in solving classical operations that can be posed on the stored data.

In this paper we present a new spatial access method, that we named *S**-tree, which extends capabilities of the S+-tree to coloured images. We first show that for practical cases the S*-tree saves up to 25% of space with respect to the S+-tree. We then prove that the S*-tree performs asymptotically the same number of disk accesses of the S+-tree to retrieve any given subset of the represented image. Finally, to assess the practical usefulness of our method, we perform experiments over an important class of queries on coloured images, namely the *window queries*. Window queries have a primary importance since they are the basis of a number of operations that can be executed in a spatial database. Window-based queries we analyze are the following:

- *exist*(*f*,*w*): Determine whether or not the feature f exists inside window w;
- *report*(*w*): Report the identity of all the features that exist inside window w;
- *select*(*f*,*w*): Report the locations of all occurrences of the feature f inside window w.

We compare our structure with the *HL-quadtree* [7], another hybrid access method combining advantages of leafcode and treecode representations, by using locational codes to represent all the nodes of a region quadtree. The HL-quadtree has been shown to be very time efficient in solving window queries. Obtained results are extremely encouraging, showing a superiority of our method both in terms of space occupancy and time performances. Since we are comparing time performances of secondary memory oriented data structures, when we state that our new data structure outperforms previous approaches for answering window queries, we refer to the fact that it reduces the number of accesses to the buckets storing the data. This is the classical *I/O complexity*.

The paper proceeds as follows. In Section 2 we briefly recall the various pixel tree (binary and quaternary) structures that have been proposed in the past for managing coloured images. In Section 3 we present our new spatial access method, namely the S*-tree. In Section 4 we give experimental results assessing the space and time efficiency of our approach, and finally, in Section 5 we present considerations for further work and concluding remarks.

2 Survey

2.1 The Bintree and the Quadtree

The *region quadtree* is a progressive refinement of an image that saves storage being based on regularity of the feature distribution. Assume we are given an image space of size $T \times T$ (e.g., pixel elements), where T is such that $T = 2^m$, containing k non-overlapping features. We proceed in the following way: at level 0 there is the whole image, of side length T. The decomposition process carried out by the quadtree recursively splits a quadrant into four equal size quadrants, until each quadrant is covered by only one feature. In the worst case, the decomposition can go on up to the pixel level, with squares of side length $\frac{T}{2^m} = 1$. The decomposition can be represented as a tree of outdegree 4, with the root (at level 0) corresponding to the whole image and each node (at level d) corresponding to a square (or *block*) of side length $\frac{T}{2^d}$. The sons of a node are, in preorder, labelled NW, NE, SW and SE. For a given image, nodes are then *homogeneous* (leaf nodes) or *heterogeneous* (non-leaf nodes). Correspondingly, we speak of homogeneous and heterogeneous blocks. Note that there exist several extensions of the region quadtree, even for representing set of overlapping images [14].

The *bintree* is the binary version of the region quadtree: the image is progressively refined alternating horizontal and vertical splits, until an homogeneous pattern is reached. Notice that in this case such a pattern is not necessarily a square. Figure 1 shows an example of an image containing 4 non-overlapping features (note that the white background is treated as a feature), along with its representing quadtree and bintree.

The bintree and the quadtree can be implemented as a tree (pointer-based representation) or as a list (pointerless representation). In the former, direct access to specific image elements is privileged, while the latter makes sequential

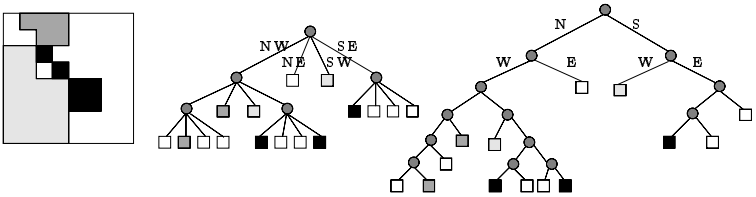


Fig. 1. Multiple non-overlapping features and their quadtree (left) and bintree (right).

access easier and simplifies disk-based representations, absolutely needed for large amounts of spatial data [10,11,13].

2.2 Secondary Memory Implementations

It should be clear from the definition that bintrees and quadtrees share a lot of properties; therefore, a secondary memory implementation of a bintree can be easily adapted to a quadtree, and vice versa. Henceforth, in the following we will freely interexchange them.

There exist substantially two categories of list-based representation of a pixel tree: the collection of the leaf nodes and the linear list resulting from a preorder traversal of the tree. One of the most attractive approaches in the first category is the *FL linear quadtree* [5] (simply *linear quadtree* in the following), introduced by Gargantini with reference to a binary image. The extension to multiple non-overlapping features is straightforward. In fact, also in this case the collection of leaf nodes can be stored as a sorted linear list, but each node now contains two fields: the *locational key*, whose digits resemble the path in the tree from the root to the node, and the *value string*, that contains the index of the feature associated with the node. Representing a pixel tree as an ordered list of the homogeneous nodes is efficient since space occupancy is reduced and performances of sequential operations are improved.

Concerning representations in the form of a linear list resulting from a preorder traversal of the pixel tree, the DF-expression [6] is surely one of the most used techniques. The DF-expression for multiple non-overlapping features can be viewed, treating the background as a feature, as a string containing two symbols: 'N', denoting non-leaf (internal) nodes, and 'L', denoting leaf nodes, followed by the index of the contained feature for leaf nodes. The representing tree is visited in preorder, and an 'N' is emitted whenever an internal node is encountered, while an 'L' followed by the index of the contained feature is emitted whenever a leaf node is encountered. As an example, suppose that the four features in Figure 1 have index 0 for the white, 1 for the light gray, 2 for the dark gray and 3 for the black. The following string is the DF-expression for the bintree in Figure 1:

$$\mathrm{NNNNNNL_0L_2L_0L_2NL_1NN\ L_3L_0NL_0L_3L_0NL_1NNL_3L_0L_0}.$$

Representing a pixel tree as a DF-expression is space efficient because of the data compression, but accessing specific blocks is time-consuming, since indexing is not provided. and this is a serious handicap for window queries processing. Therefore, an implementation based on B^+-trees for a linear quadtree representation is straightforward [1], while it is not possible for a DF-expression.

A first step towards the integration of leafcode and treecode representations has been done by de Jonge et al. [3], who defined a secondary memory representation of binary images named *S^+-tree*. The S^+-tree is obtained in the following way: preliminary, we visit in preorder the pixel tree, emitting a '0' ('1') when an internal (leaf) node is encountered. The bitstring thus produced is called a *linear bintree*. An additional bitstring, called *colour table*, records the colours of the leaves in preorder, by letting a '0' ('1') represent a white (black) leaf. The two bitstrings thus obtained are named *S-tree*. The S^+-tree is then built by indexing with a B^+-tree a list of data pages containing a segmented and augmented S-tree representation of the image. Each data page constitutes a self-contained local S-tree that can be searched independently. A data page consists of a bitstring merging the linear bintree (which grows from the beginning of the page) and the colour table (which grows from the tail of the page) of the local S-tree. Between them there is some unused space (actually, this unused space is negligible for binary images but it is not for coloured images, as we shall see in the next section). Moreover, at the very beginning of the page there is a *linear prefix* which can be regarded as the summary of all the data pages preceding the actual one. This linear prefix is determined in the following way: when a data page becomes full during the building process, a new page is created and a *separator* between the pages is stored in the index. Such a separator is built by encoding the path from the root of the bintree to the node which caused the filling of the page, emitting a '0' when moving towards left, a '1' otherwise. Since it is imposed that the last node stored in a page must be a leaf (we will analyze this constraint more in detail in the next section), it follows from the preorder visit properties that the last bit of a separator is always a 1[1]. Consequently, the linear prefix is built by encoding with a '0' a 0 in the separator, and with a '01' a 1 in the separator. The 0 added before the 1 actually represents a *dummy leaf*, staying for a left subtree (stored in a previous page) along the path to the node which caused the filling. The linear prefix therefore provides the information needed to retrieve a node in a page, since it resembles the whole bintree preceding the nodes in such a page by condensing all the left subtrees in leaves.

3 The S*-Tree

In the previous section, we mentioned that a tight constraint during the process of building the S^+-tree is that the last node stored in a page must be a leaf. There are several convincing reasons to do that for binary images:

[1] In fact, if the last stored node is a left leaf, then the node which caused the filling is a right leaf (its sibling), while if the last stored node is a right leaf, then the node which caused the filling is some right son of an ancestor of the right leaf

1. Since the last node is a leaf, by preorder visit properties it follows that the first node on the next page is a right son, and therefore the separator between the pages will end with a 1. This is important, since it allows to store the separators using only $2m$ bits, where m is the resolution of the image, without encoding the depth of the node the separator refers to which.
2. Since for binary images no information is associated to internal nodes (they are simply gray), we have at most $2m-1$ unused bits per page. Considering that a page is generally 512 bytes in size and that a reasonable upper bound on m is 16, it follows that we waste in the worst case less than 1% of space.

However, the latter observation does not hold any more for coloured images. In such a case an internal node has an amount of information associated with it: more precisely, to each internal node we have to associate a *colour string* of k bits (where k is the number of features contained in the image), in which the ith bit is 1 iff the node contains the ith feature. In fact, associating a colour string to internal nodes greatly improves the performances in executing several spatial operations [7]. Thus, if $m = 16$ and $k = 32$, the wasted space could be as big as 124 bytes, i.e., about a 25% of the page size! Therefore, it is clear that for coloured images we have to abandon the constraint that the last node stored in a page must be a leaf node. The question is: can this be done without modifying the separators, i.e., without augmenting the space used for the index? The answer is yes, on condition that a small overhead is paid in terms of the time spent when a search to a given node is performed. In fact, a problem arises letting the last node stored inside a page to be internal, that is, it fails the statement that the last bit of a separator is always a 1. This is because the node which caused the filling could be a left son, and iteratively its parent could be a left son and so on. Therefore, in the separator, after the rightmost 1, there could be some meaningful 0s (actually, as many as $2m-1$), i.e., 0s that effectively lead to the node which caused the filling. Does this affect the search of a given node through the structure? Only to a small extent, as the following theorem states:

Theorem 1. *Let $\ell = 2m$ be the length of the index keys in the B^+-tree storing the S^*-tree, and let $\pi(x) = \{0,1\}^t$ with $t \leq \ell$, be the path from the root to a node x to be retrieved in the S^*-tree. Then, as soon as each page in the B^+-tree contains at least ℓ nodes of the bintree, it follows that at most two contiguous pages in the B^+-tree must be visited to retrieve x.*

Proof. We start by noting that the assumption that each node in the B+-tree contains at least ℓ nodes of the bintree is not restrictive in applicative cases: for example, for $m = 16$ and $k = 32$, it suffices to fix the page size of the B+-tree to 128 bytes.

Let $\pi_i \in \{0,1\}$, $i \leq \ell$ be the ith bit of $\pi(x)$ and let π_r be the rightmost 1 of $\pi(x)$. We can therefore write $\pi(x) = \pi_1 \dots \pi_r \pi_{r+1} \dots \pi_t$, with $\pi_{r+1} = \dots = \pi_t = 0$. To retrieve x, we will search in the B+-tree for the key $k_x = \pi_1 \dots\dots \pi_r \pi_{r+1} \dots \pi_\ell$, with $\pi_{r+1} = \dots = \pi_\ell = 0$. Let k_a be the key in the B+-tree reached by searching k_x and let P_1, P_2 be the two pages separated by k_a. Without loss of generality, let us assume that $k_a \leq k_x$. We will show that x

must be either in P_1 or in P_2. Notice that k_a represents a separator, i.e. a node in the associated bintree, say a, having a path $\pi(a)$ from the root. Of course, $\pi(a) \leq k_a$. Two cases are possible: $k_a < k_x$ or $k_a = k_x$.

The former case is trivial. In fact, if $k_a < k_x$, then in a preorder visit, a must precede x, i.e., $a \prec x$, from which it follows that x must be in P_2.

Let us now analyze the latter case, i.e., $k_a = k_x$. Remember that $\pi(a)$ is the path to the first node stored in P_2. To establish the thesis, we have to prove that x cannot be stored in any page preceding P_1. We start by noting that k_x does not only represent the sequence $\pi(x)$, but also all the sequences of the following set:

$$S = \{\sigma \in \{0,1\}^s | \sigma = \pi_1 \dots \pi_r \pi_{r+1} \dots \pi_s, \pi_r = 1, \pi_{r+1} = \dots = \pi_s = 0, r \leq s \leq \ell\}.$$

Notice that $|S| = \ell - r \leq \ell$ and that $\pi(a), \pi(x) \in S$. If x is stored in a page preceding P_1, then for any node y stored in P_1, it will be $x \prec y \prec a$, from which it follows that $\pi(y) \in S$. This means, all the nodes in P_1 have a path belonging to S. But this is a contradiction, since P_1 contains at least ℓ nodes and $|S \setminus x| \leq \ell - 1$. □

The above result guarantees that the only delicate case to be managed is when the returned key from the searching in the B$^+$-tree equals the key we are looking for. In this case, we will load in main memory both the pages pointed by such a key, thus performing an extra access on secondary memory. This scenario is quite unlike to happen, and therefore we conclude that our approach works well for all practical purposes.

We finally remark that we choose in our design of the S*-tree to eliminate the linear prefix from the pages, since it can easily be recomputed from the separators in the B$^+$-tree. This will add a small overhead in terms of CPU time, but, on the other hand, we will reduce the space occupancy and simplify the standard B$^+$-tree merging operation: in fact, when two pages of the B$^+$-tree are merged together as a consequence of an underflow, the separator in the B$^+$-tree must be changed, and so for the linear prefix inside the page. This can produce a time expensive shifting of all the bits inside the page. Eliminating the linear prefix will eliminate this problem. The actual layout of a page of the S*-tree is given in Figure 2. Note that the free space will be at most k bits. The *tree pointer* points to the next available position in the linear tree stack, the *colour pointer* points to the next available position in the colour string stack while *next* is a pointer to the next page in the sequence set [3]. The field *length* stores the length of the separator.

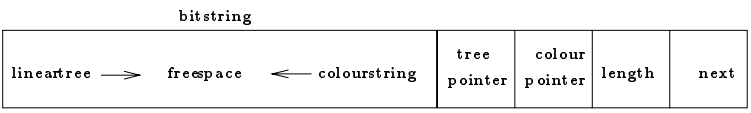


Fig. 2. Layout of a page of the S*-tree.

Figure 3 provides the complete B+-tree containing the bintree of Figure 1. Note that we set the size of the bitstring to 36 bits. With any external node we have associated a colour key: we encoded the white feature with '00', the light gray with '01', the dark grey with '10' and the black with '11'. For internal nodes, the associated colour string has 4 bits associated, from left to right, to white, light gray, dark gray and black. A bit in the colour string is set to 1 iff the associated feature is contained in the subimage individuated by the node. The two separators of the resulting three pages are 00001 and 001110, respectively. Thus, the second separator will be ambiguous, since its last digit is a 0. For example, looking for the node 00111 will retrieve the key 001110 from the B+-tree. As proved above, in this case we will not visit only the page following the retrieved key; instead, we will preliminary visit the page preceding the key: we compute the linear prefix by using the key 000010 and the length 5 stored in the page (thus the separator will be 00001 and the linear prefix will be 000001, since we codify a '0' with a '0' and a '1' with a '01'). Using the linear prefix, we are then able to retrieve the node 00111 as the last one of the second page (see [3] for details).

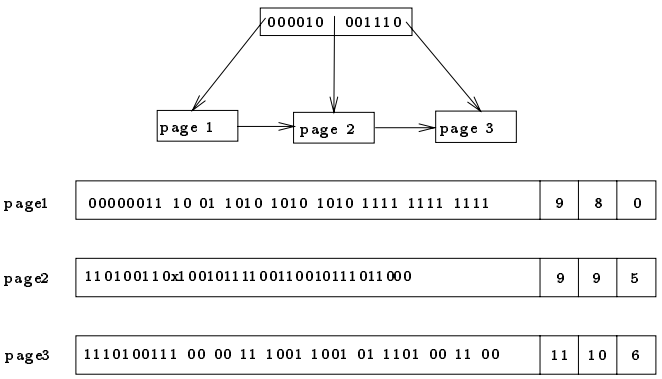


Fig. 3. The resulting B+-tree storing the S*-tree.

4 Experimental Results

In this section we present detailed experiments comparing the S*-tree with the HL-quadtree, since this latter spatial access method for coloured images has been shown to be very efficient with respect to other linear quadtree implementations [7]. We implemented both methods in C language and run the experiments on a SUN SPARC workstation with UNIX operating system. We executed the window queries on a set of images of size $2^{10} \times 2^{10}$ containing multiple non-overlapping features, ranging from satellite views to landuse maps. For the sake of brevity, we here present results for a set of 10 images containing 32 features and consisting of meteorological satellite views of North America. Figure 4 shows a sample image.

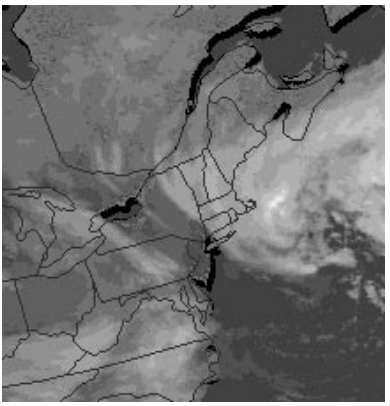


Fig. 4. A sample image (North America) containing 32 features.

We considered the following window queries, of primary importance for multiple non-overlapping features [2]:

- *exist*(*f,w*): Determine whether or not the feature f exists inside window w;
- *report*(w): Report the identity of all the features that exist inside window w;
- *select*(*f,w*): Report the locations of all occurrences of the feature f inside window w. This means to output all blocks homogeneous for feature f.

The basic approach to process all window queries was to decompose the query over a window into a sequence of smaller queries onto the maximal blocks contained inside the window [2]. Given a square window query of side n, these maximal blocks are $O(n)$ [4] and can be determined in linear time [8]. A detailed description of the algorithms used to process the queries can be found in [7]. For our purposes, it suffices to recall here that the *exist*(f, w) and the *report*(w) queries can be answered in $O(n \log_r T)$ I/O time, where r is the order of the B$^+$-tree, while the *select*(f, w) query can be answered in $O(n \log_r T + n^2/r)$ I/O time [7]. In our experiments, the page size of the storing B$^+$-tree was fixed to 512 bytes, while the order r has been fixed to 16.

4.1 Space Usage

A first comparison has been made on the space used by the HL-quadtree and the S*-tree. To this aim, we let the resolution of the images change from $2^8 \times 2^8$ to $2^{10} \times 2^{10}$. Figure 5 shows the results. From the drawing, it emerges that the S*-tree uses about 1/4 of the space used by the HL-quadtree. Therefore, the improving is substantial. This positively influences time performances, as we shall see in the next section.

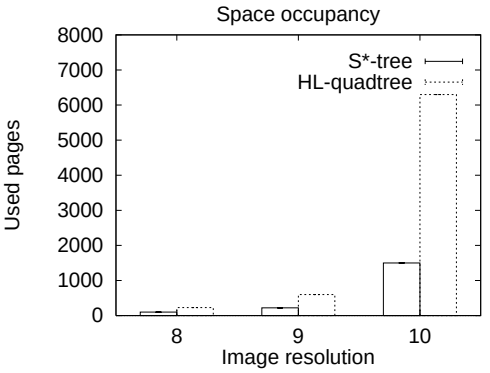


Fig. 5. Space occupancy comparison between the two methods.

4.2 Time Performances

To analyze the time performances of the HL-quadtree and the S*-tree, we used the classical measure of I/O complexity, that is, the number of disk accesses on secondary memory. The CPU time is indeed negligible with respect to the time spent in retrieving a page on secondary memory. In the following, we make the standard assumption that each secondary memory access transmits one page of data (a *bucket*), and we count this as one operation. We tested the two methods for each of the window queries and for each of the considered images. We randomly generated the anchor of 100 square windows of side $n = 50 \cdot i, i = 1, \ldots, 8$ (i.e., a total of 800 query windows). The windows have been "wrapped around" the image space whenever they extended beyond the borders of the image. Then, we computed the number of disk accesses for solving the queries using the two approaches, for each of the images, and we computed the arithmetic mean.

Concerning the *exist*(f, w) query, we focused on two different densities of feature distribution, namely features covering about 5% and 30% of the image space, respectively. We found such features for all the images, since the images were statistically similar. Figure 6 shows the results. From the drawings, we derive that in both cases there is approximately a 20% of saving in the number of accesses using the S*-tree. It is worth noting that as soon as the feature density increases, the I/O complexity decreases dramatically, and even though the theoretical I/O complexity depends on the window side, the query is solved in much less time. In fact, in case of high density, the probability of finding the feature after few accesses is very high.

Concerning the *report*(w) query, Figure 7(a) shows the results. Here, the I/O complexity of the two methods increases as soon as does the window side, since all the maximal blocks of the window need to be examined. In general, the S*-tree answers the query by doing about 20% less of the I/O accesses done by the HL-quadtree.

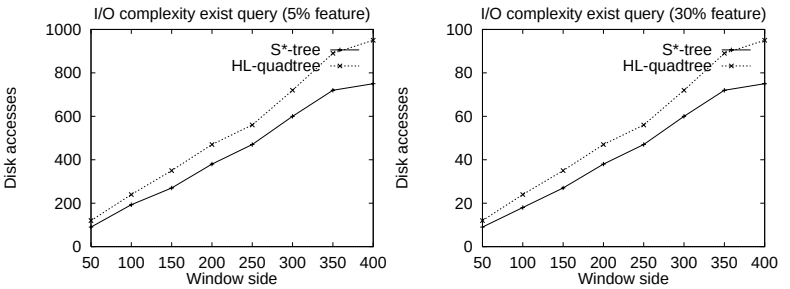


Fig. 6. Time performances of the two methods for the *exist*(f, w) query: (a) feature covering 5% of the image space; (b) feature covering 30% of the image space.

Finally, concerning the *select*(f, w) query, we focused on the feature covering about 30% of the image space. As for the *report*(w) query, the I/O complexity of the two methods increases as soon as does the window side, since also in this case all the maximal blocks of the window need to be examined. Furthermore, we here have a larger number of disk accesses, since to return all the occurrences of a feature we need to examine all the descendants of a node corresponding to a given maximal block. Figure 7(b) presents the results, showing that the S*-tree answers the query using about 70% of the I/O accesses used by the HL-quadtree. Therefore, we have up to 30% of time saving.

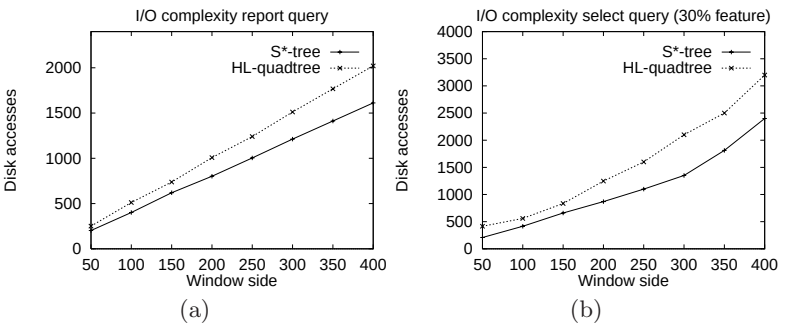


Fig. 7. Time performances of the two methods: (a) *report*(w) query; (b) *select*(f, w) query.

5 Conclusions

In this paper we have proposed and analyzed the S*-tree, a new time and space efficient disk-based representation of images containing multiple non-overlapping features. We used as time performance measure the number of secondary storage accesses for solving the classical window queries, and our experiments showed that the new approach outperforms a previous efficient spatial access method proposed in literature, namely the HL-quadtree [7]. More precisely, saving in time and space is about 20%.

Future work will be in the direction of an extension of this new encoding technique to the more general case of images containing multiple overlapping features. We also plan to test the S*-tree in performing other spatial operations.

References

1. W.G. Aref and H. Samet. A B+-tree structure for large quadtrees. *Computer Vision, Graphics and Image Processing,* 27(1):19–31, July 1984. 160
2. W.G. Aref and H. Samet. Efficient processing of window queries in the pyramid data In *Proc. of the 9th ACM-SIGMOD Symposium on Principles of Database Systems,* pages 265–272, Nashville, TN, 1990. 164, 164
3. W. de Jonge, P. Scheuermann, and A. Schijf. S+–trees: an efficient structure for the representation of large pictures. *Computer Vision, Graphics and Image Processing: Image Understanding,* 59(3):265–280, May 1994. 157, 160, 162, 163
4. C. Faloutsos, H.V. Jagadish, and Y. Manolopoulos. Analysis of the n-dimensional quadtree decomposition for arbitrary hyperectangles. *IEEE Transactions on Knowledge and Data Engineering,* 9(3):373–383, 1997. 164
5. I. Gargantini. An effective way to represent quadtrees. *IEEE Trans. on Pattern Analysis and Machine Intelligence,* 25(12):905–910, 1982. 157, 159
6. E. Kawaguchi, T. Endo, and M. Yokota. Depth-first expression viewed from digital picture processing. *IEEE Trans. on Pattern Analysis and Machine Intelligence,* pages 373–384, July 1983. 157, 159
7. E. Nardelli and G. Proietti. Time and space efficient secondary memory representation of quadtrees. *Information Systems,* 22(1):25–37, 1997. 156, 158, 161, 163, 164, 164, 167
8. G. Proietti. An optimal algorithm for decomposing a window into maximal quadtree blocks. *Acta Informatica, 1999.* To appear. 164
9. H. Samet. The quadtree and related hierarchical data structures. *Computing Surveys,* 16(2):187–260, June 1984. 157
10. H. Samet and R.E. Webber. A comparison of the space requirements of multidimensional quadtree-based file structures. *Visual Computer,* 5(6):349–359, December 1989. 159
11. C.A. Shaffer and P.R. Brown. A paging scheme for pointer-based quadtrees. In D. Abel and B.C. Ooi, editors, *Advances in Spatial Databases*, pages 89–104. Lecture Notes in Computer Science 692, Springer Verlag, 1993. 159
12. M. Tamminen. Encoding pixel trees. *Computer Vision, Graphics and Image Processing,* 2:174–196, 1984. 157
13. M. Vassilakopoulos and Y. Manolopoulos. Analytical comparison of two spatial data structures. *Information Systems,* 19(7):269–282, 1994. 159

14. M. Vassilakopoulos, Y. Manolopoulos, and K. Economou. Overlapping quadtrees for the representation of similar images. *Image and Vision Computing,* 11(5):257–262, 1993. 158

Workflow Management System Using Mobile Agents

Zoran Budimac, Mirjana Ivanović, and Aleksandar Popović

Institute of Mathematics, Faculty of Science, University of Novi Sad,
Trg D. Obradovića 4, 21000 Novi Sad, Yugoslavia
{zjb,mira,ror}@unsim.ns.ac.yu

Abstract. The paper describes an infrastructure for implementation of workflow management system using mobile agents. The usage of mobile agents in modelling and implementation of a workflow is a novel approach and simplifies the workflow management. Besides, the suggested infrastructure introduces fresh ideas in the field of mobile agents as well.

1 Introduction

Using the workflow management system is a modern trend in modelling and implementation of an information flow and business processes in a company [6,10]. Well organized and fully implemented, a workflow can replace a large part of a classical information system of a company. In a focus of a workflow is "work" (information, a task, a document, announcement, data, etc.) that flows through different points in a company. Deadlines and conditions of a work transition are defined in order to achieve the flow of work. When a condition is met, work is transferred to a next stage, where the next condition has to be fulfilled. At the end of its itinerary, the work is done.

According to most general definition, a mobile agent is a program that is able: to stop executing at one node in a computer network; to transfer itself to another node in a network; and to continue execution there. More precise definitions include additional requirements for a piece of code to be a mobile agent. All definitions however, stress the important feature of mobile agents - autonomous behaviour. A mobile agent autonomously decides when and where will be transferred. Mobile agents are a very fresh research area in the field of a distributed programming and artificial intelligence [7,9,12,13,14,15,16,18]. A mobile agent is today often an object of some object-oriented programming language. Advantages of mobile agents with respect to classical techniques of distributed programming are numerous. Among many, we stress the following ones:

– Potentially better efficiency of the whole system. A client program goes to a server-node, directly communicate with a server program, and returns to an original node with a result. Overall network traffic is thus potentially reduced, because all intermediate network traffic is now unnecessary.

Eder et al. (Eds.): ADBIS'99, LNCS 1691, pp. 168–178, 1999.
© Springer-Verlag Berlin Heidelberg 1999

– Greater reliability, because the connection between nodes must not be established all the time.

In this paper, data structures, agents, tools, and principles for implementation of a workflow management system using mobile agents are presented. This approach has several main advantages over more classical approaches in organization and implementation of workflow management systems:

– Uniformity of organization and implementation. Mobile agent system is used for implementation of: a workflow administration and monitoring tools, workflow client applications, and workflow enactment services (Fig. 1)[10]. Moreover, agents are used as a uniform interface to invoked applications and other workflow enactment services. The other approaches in organization and implementations of workflow management systems often use different tools and techniques to implement different parts of a system.

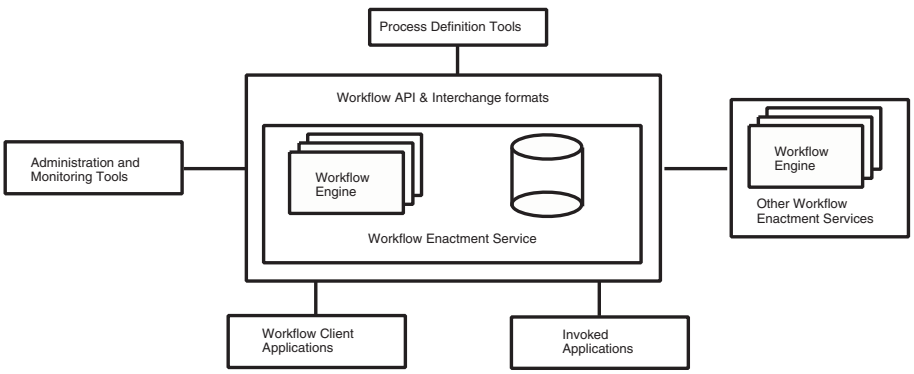


Fig. 1. The Workflow Reference Model

– Simpler flow management. Mobile agents are work-items that are passed to different users and autonomously take care of their current position and further itinerary. Other approaches often use separate mechanisms to send work-items (transactions, e-mail messages, etc.) and to record their current position and status (centralized control, database.)
– Flexibility. The flow of work is easily changed when using mobile agents. It is done only by changing agent itineraries or by introducing new agents. Organizational structure of a company is contained implicitly in agent itineraries and is easily changed as well. In other approaches, it is necessary to change central administration to achieve changes of flow or organizational structure.
– Scalability. The workflow implemented with mobile agents can easily grow with the underlying company or complexity of its business processes. In those cases, just more agents are added, without the need to change and understand the rest of the system. Other approaches request changes in most parts of the workflow management system.

– Support of unstructured business processes. With the use of mobile agents, business processes need not be well structured.

The system we propose in this paper consists of individual agents with autonomous behaviour. It is even more uniform, decentralized, flexible, and scalable than other workflow management systems with mobile agents proposed recently [4,5]. This system also stresses that mobile agents are (or can be) a separate programming style (paradigm), rather than only a technical improvement of a classical distributed programming paradigm.

The rest of the paper is organized as follows. In the next section, an abstract class *Task* is described. It is a basis for creation of individual agents representing works (work-agents.) A work-server that hosts work-agents is introduced in the third section. In order to ease the application of the workflow system and to increase its reliability, it is necessary to implement additional tools and specialized administrative agents. These two aspects of a proposed workflow system are discussed in sections 4, 5, 6, and 7. In the section eight, a reliability and protection of the proposed system are discussed. In the ninth section, the related work is presented, while the tenth section concludes the paper.

2 Class *Task* and Work-Agents

Class (in the sense of object-oriented programming) *Task* represents an abstract (i.e., every) work in the proposed workflow system. This class is a descendant of a class that represents a mobile agent in a chosen mobile agent system. Objects of the class *Task* thus become mobile as well. The class contains the following attributes and methods (Fig. 2):

– attribute containing work identification,
– attribute containing work deadlines,
– attribute containing an owner of work (the person that created the work),
– method for work externalization (to save the work into a file),
– method for work internalization (to load and recreate the saved work),
– itinerary,
– method for presenting a user-interface.

The itinerary is a list of triples of the following form: (*node, condition, methods*). It represents a flow of work-agent through a network. A *node* represents an address of a node where the work will transfer itself from the current node. Only methods enlisted in the list *methods* are active on the current node. The work-agent will transfer itself to the *node* when and if the *condition* (a logical function) is fulfilled. If an itinerary of a work-agent can contain alterative routes through a network, then the itinerary must be represented as a tree, rather than as a list.

Work-agent classes are descedants of the class *Task* and contain attributes that describe a concrete work and methods that can be used to process the work. For example, the work "paper writing" (class *Paper*) adds an attribute

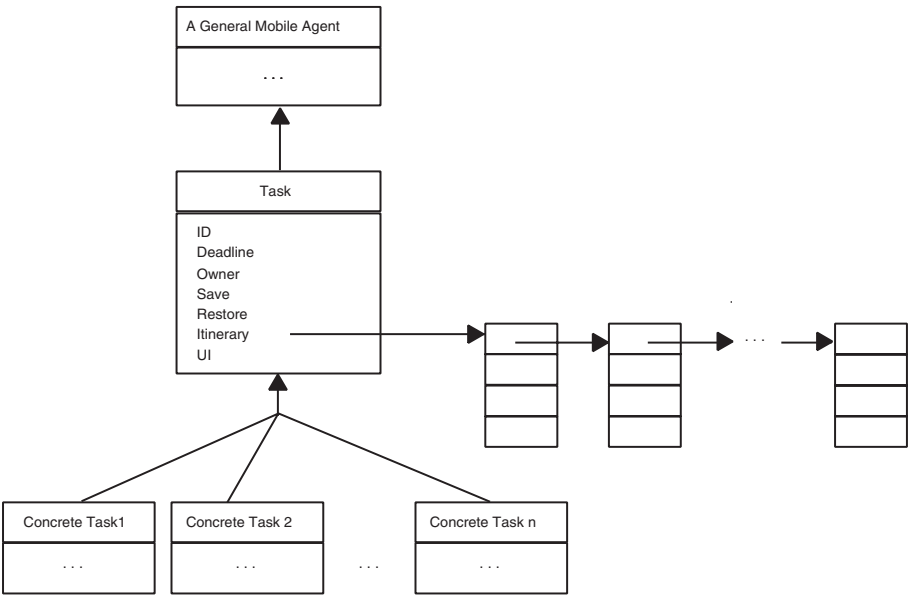


Fig. 2. The class *Task* and its descedants

that contains the file with a paper and the method that will be used for paper editing (i.e., a text-processor.) The work "job" (class *JobForm*) adds attributes of a form and adds methods for filling the form entries.

3 Work-Server and Work-Host

Our system now consists of work-agents that represent concrete tasks and are created by inheriting an abstract class *Task*. Work-agents are mobile because the class *Task* inherits the class that describes a mobile agent of underlying mobile agent system. Besides work-agents, a work-server is needed as well. Work-server is a program that executes all the time on every node in the workflow system. Its basic tasks are the following:

- Listens the designated port, waits for incoming agents, internalize them, puts them into a list of agents on that host, and activates them.
- Inform the user about an arrival of new work.
- For every work-agent periodically (for example twice in an hour), calls the function *condition* for the current node.
- If the work-agent is too long on the same node, alerts the user directly or using some existing service (e-mail, for example).
- At the end (before switching off the computer), externalizes all agents that are currently under its supervision.
- At the beginning of its execution (after the computer is switched on), internalizes all saved agents.

The work-server must be kept small and simple. That way its execution will not significantly reduce overall node performance. Because of that, all other necessary administrative functions are implemented as separate, specialized agents. Some of enlisted work-server's tasks can be done by other agents, as well. For example, the work-agent itself can notify the user about its arrival and can alert the user if it is too long on the same node. Note that every work-agent delivers itself to the next node, if the *condition* is met. It is not the responsibility of the work-server - it just calls *condition* periodically.

Besides work-server, every node contains one work-host that can be implemented as a stationary agent. The role of work-host is to offer the user basic data about all work-agents that currently reside on the node. The host-agent will enable the user to choose one of the work-agents. It is invoked explicitly, after which it establishes connection between the user and all residing work-agents. The work-host is very simple - it just uses appropriate work-agent attributes and invokes its methods. For example, to enable the user to work with some work-agent, work-host will invoke a user interface of that work-agent. A user interface will then present all available possibilities, from the list *methods* in the itinerary.

Work-agents, work-servers, and work-hosts are the only software components that are really necessary for the workflow system. However, if the system is to be more user-friendly, more reliable, and more flexible, additional tools and specialized agents are needed. In the next several sections some possible additions will be introduced.

4 Templates and Template Library

A work-agent is defined by inheriting the class *Task* and by addition of new attributes and methods. The most important part in a definition of a new work-agent (at least from the workflow point of view) is a definition of an itinerary. For every class of a work-agent, a template can be defined. The template for some work-agent class is an object of that class with fixed itinerary that contain empty *node*s. Besides partially instantiated itinerary and possibly some other attributes, most of other attributes are not instantiated.

For example, the template for *JobForm* will contain the internal representation of the form, and the itinerary with three entries: a) the new employee, b) his/her chief/supervisor, and c) employee records (database). In this itinerary, the node of a new employee and the node of his/her chief are un-instantiated.

All available templates are organized into a template library. A user creates a new, concrete work-agent by picking a template from the library and instantiating undefined attributes and nodes in the itinerary. The work-agent will then autonomously transport itself to the first node in its itinerary.

5 New Work-Agent Classes

In the real companies, it is not possible to define all possible work-agents and their templates in advance. New work-agents may be needed on a daily basis. To employ workflow system in a real working environment, it should be possible to create brand-new work-agents by the end-user of the workflow system. This could be done by defining and implementing a software tool that enables the visual definition of a new work-agent and its template. An alternative is to use an existing tool for process modelling and then to extract necessary data.

New attributes would be identified and their types would be picked up from the list of predefined data types. The common methods would be picked up and somehow combined from the repository of predefined methods and procedures. Simpler methods could be created by using some visual programming language/environment. Another possibility is to use PLES - simple yet powerful programming language [11]. Source PLES programs can be embedded into an application software program and executed from there. PLES programs can be freely created and changed by the end-user.

The itinerary in the template would be created graphically by connecting nodes in a predefined, graphically represented computer network. Conditions for agent transport would be defined in a similar manner as noted in a previous paragraph.

Because the creation of a new work-agent involves at least some knowledge in procedural thinking (if not programming), new work-agents would be created by designated users with needed background knowledge.

Based on graphical representations of new work-agents and their templates, a pre-processor would create textual representation of new class. The class would be afterwards compiled and put into the system.

6 Access to Internet and Databases

The suggested workflow system works only on nodes where work-servers and work-hosts are installed. In the real company environments there will be a lot of business processes that cannot be completed without access to nodes outside of the implemented workflow system. In those cases, an access to common Internet services is needed.

More precisely, we need special (stationary) agents that are able to send and read e-mail messages, transfer files from one node to another using FTP protocol, access web-pages from other nodes etc. With existence of these Internet agents, every work-agent can access the service it needs. For example, the work-agent *Paper* can at the end of its itinerary send by e-mail the final version of the paper to the designated target.

Furthermore, work-agents will often have to access the internal or external databases, to retrieve or store data. For that purposes, a specialized database agent is needed. It can be mobile and can be created and sent by any work-agent in the system to perform its own itinerary. Most commonly, the itinerary would

consist of going to the database server, perform the query, store the result into itself, and going back to the sender. Depending on the type of the database (internal with known organization and implementation, or external) and the current accessibility of the network, such a database agent can also decide to: a) stay stationary and send the query in a standard way, or b) to use Internet service to access an external database.

7 Other Specialized Agents

The workflow system we suggest is fully distributed, without central administration, control, and maintenance. All reports, control, and management are achieved by creating and sending specialized agents that will communicate with other agents in the system and achieve the intended results. In this section, we enumerate some possible specialized agents.

7.1 Automated Work-Agents

Besides work-agents that are explicitly created by the user, work-agents that will be created automatically are often needed. They would be automatically created at designated time and they would transfer themselves according to their predefined itinerary. Classical examples are reminders for periodical activities.

For these purposes, the work-server is extended by a new function - creation of a work-agent if certain conditions are met.

7.2 Trackers

A user of a workflow system can send a tracker-agent at anytime. It will search for all agents owned by the user on every node in the system, and gather a report. Once back, it will present the report of the status and position of all found agents.

7.3 Detour Agents

If a user (or its node) is not able for a longer period of time to fulfill its duties, all agents aiming to that node have to detour. It is done by changing their original itineraries with alternative ones. A detour-agent carrying a new itinerary is sent to search for other agents to replace their old itineraries with the new one.

A detour-agent can be implemented in several ways. For example, every work-agent can have its own detour agent. A new itinerary in such a detour agent is defined using the tool described in the fifth section. A detour-agent is then sent to all nodes to look for all agents that belong to certain class. When such an agent is found, a detour agent will replace its itinerary with the new one.

Another possibility is to create a general detour-agent that is able to change the itinerary to all possible work-agents. It is easily achieved if it is only needed to change one node in work-agent itineraries with an alternative one.

7.4 Advisors

If an itinerary contains alternative routes, advisor-agents stationed at some nodes could direct incoming work-agents to take other route. They would be created automatically if some conditions are met (for example, if there are already too many waiting work-agents on that node.)

8 Protection and Reliability

From the user point of view, all agents in the system (work-agents and specialized agents) behave autonomously. They decide on their own, when and where to go, and what they will do on the new location. Sometimes, their behaviour is determined when they are created. Often, the behaviour is influenced by communication with other agents - detour-agents or advisors.

Under these circumstances, the only responsibility of the user is first to activate the work-agent and afterwards to choose all actions an agent provides at that time. The responsibility for other activities lies on the agent itself. To implement this concept, the workflow system has to be secure and reliable.

8.1 Protection

Enforcing protection of a mobile agent system usually means: a) to protect agents from malicious nodes, and b) to protect nodes from malicious agents. Although protection is an important issue in any mobile agent system, we feel that in a closed system such as ours, all nodes can be regarded as trusted. The only way the agent communicates with foreign nodes and services, is via Internet services. Therefore, we neglect the issue of protection of agents from malicious hosts and a protection of hosts from malicious agents. This problem should be addressed when work-agents from our system are allowed for transport to foreign nodes, and when foreign agents are allowed for access to nodes of our system.

It is only needed to take care of the user's rights to create, access, and change agents in the system, as well as of the agent rights to access other agents and operating system services. Most commonly, rights of an agent can be analogous to existing rights of its owner. Users' rights with respect to agents must be established separately, as a part of a workflow system.

8.2 Reliability

Reliability of the mobile agent system is most often regarded as enforcing the "exactly once" semantics [17]. An agent cannot disappear on its way (because its current node is malfunctioning) nor it or its operations cannot be multiplicated. The reliability of this kind is most commonly solved by generating spare copies of every agent ("shadows" [2] or "rear-agents"). Shadow follows the migration of its original and replaces it when and if original disappears. Besides this, it is useful if all agents on every node are externalized periodically.

Many mobile agent systems already have built in mechanisms for protection and for reliability. If it is not the case with chosen mobile agent system, basic protection and reliability mechanisms must be enforced separately by the workflow system.

9 Related Work

Several authors have recently suggested a usage of mobile agents in workflow management and electronic commerce [1,4,5]. Rather than going top-down in describing possible use of mobile agents in workflow management, this paper takes bottom-up approach. Our system is highly decentralized and consists solely of individual agents with autonomous behavior, which differentiates it from approaches in [4,5]. The only centralized control is the control of user rights to create, access, and change agents and templates. This paper extends ideas given in [3].

With respect to decentralization, our system resembles [19], that is based on static CORBA objects. While decentralization in [19] was one of explicit design goals and had to be explicitly implemented, decentralization in our system comes for free as a natural consequence of agent mobility and autonomous behavior. Moreover, our system is uniform - it is designed to use only one mechanism (mobile agents), without the need for additional mechanisms (transactions, HTTP protocol, HTML documents, Web browsers, CORBA, etc. [19].)

The proposed workflow system brings some fresh views in particular fields of a workflow management and in mobile computing. In both fields, the advantages of highly decentralized and disributed approach in designing a system, have not been often recognized. The most cited advantages of mobile agents are associated with better performances and a higher reliability of distributed systems. Our workflow system emphasizes the fact that mobile agent has organizational advantages as well. Mobile agent systems can be regarded as a separate programming paradigm and not only as an improvement of distributed programming style. Solutions to some problems are easier to program, understand, and maintain, if implemented using mobile agents.

The similar holds for workflow management systems - the use of mobile agents frees the workflow management system of centralized control that is typically the most complicated part of the system.

According to [8], most applications of mobile agents still belong to client/server software architecture. With respect to classical client/server software systems, in a mobile system a client can transport itself nearer to the server-node or even to the server-node itself. The division between service providers and service users still holds however. In our workflow management system there is no such distinction. A work-agent represents "work" to be done on different nodes in a network, and it is both a provider of some service and a client of other services.

Most agents suggested in our system (especially work-agents) are not mobile all the time. They in fact spend most of their lifetime waiting to be chosen by the

user. Agents in other systems are mostly executing all the time. The need to wait on some nodes, require new features of agents. For example, externalization of agents in other systems is a useful feature, while in our system it is a key feature.

In a proposed system, conditions for agent transport to the next node are explicitly enumerated in the agent's itinerary. The creation, understanding, and maintenance of agents are therefore simpler. In most of other systems, conditions for transport are embedded in a monolithic agent body.

10 Conclusion

The main characteristics of a workflow system suggested in this paper are almost full decentralization and distribution of workflow functions. Networked workstations are the necessary hardware infrastructures for such a system.

The proposed organization mimics usual user activities in a real flow of work. Moreover, it relieves them (or any centralized control) from the need to know what to do next with the work-agent. Every user takes care only of work-agents that are currently on its node. Where they came from, why they are here, and where they will go later, is not concern of the user.

Since the system consists of many autonomous agents, the system is easily changed, extended, and improved. It is often needed just to introduce new agents, without the need to change and even to understand the rest of the system.

The organization and implementation of a proposed workflow management system are also easy to understand and follow, because most of its parts are uniformly implemented as (mobile) agents. For example, administration and monitoring tools are implemented as special agents: trackers, detour-agents, advisors, etc. The role of workflow enactment service is done by work-servers, work-hosts, and underlying mobility of work-agents. Process definitions are embedded into templates and therefore in itineraries of individual agents.

A fully functional workflow system for middle sized companies consists of several dozen agent templates and several hundred agents migrating through the network. Such system typically consists of more complex work-agents than those mentioned here. For example, they can employ work-agents that create many children, send them to different destinations and wait for them at some place.

Full implementation of the proposed system requires a lot of work on different part of the system. However, the basic system consisting of several work-agents, a work-server, and a work-host, can be achieved in a few months time. It could be immediately used, and then gradually extended with new agents. The implementation of the proposed workflow system has begun using IBM's Aglets.

Acknowledgments

We are grateful to Mrs. Sonja Stevanović and Mr. Zvonimir Dudan from "Energosoft", for their views and discussions about workflow implementations and for their willingness to use and test the system in "Energosoft."

We thank the referees for their constructive comments that helped us to improve some aspects of the paper.

References

1. Ambroszkiewicz, S., Cetnarowicz, and Radko, B.: Enterprise Formation Mechanism Based on Mobile Agents. Technical Report, http://www.ipipan.waw.pl/mas 176
2. Baumann, J. And Rothermel, K.: The Shadow Approach: An Orphan Detection Protocol for Mobile Agents. Bericht 1998/08, Stuttgart University, Faculty of Informatics, 1998. 175
3. Budimac, Z., Ivanović, M., and Popović, A.: An Infrastructure for a General, Highly Decentralized Workflow Using Mobile Agents. Position Statement, Proc. of CEEMAS '99, Moscow, Russia, in print, 1999. 176
4. Cai, T., Gloor, P.A., Nog, S.: Dartflow: A Workflow Management System on the Web using Transportable Agents. Technical Report PCS-TR96-186, 1996. 170, 176
5. Chang, W., Scott, C.: Agent-based Workflow: TRP Support Environment. Computer Networks and ISDN Systems, vol. 28, issues 7-11, 1997. 170, 176
6. Debenham, J.: Constructing an Intelligent Multi-Agent Workflow System. Proc. of AI '98, Brisbane, Australia, pp. 119-166, 1998. 168
7. General Magic: Mobile Agents White Paper. General's Magic homepage, 1997. 168
8. Gray, R. S.: Agent Tcl: A Flexible and Secure Mobile Agent System. PhD Thesis, Dartmouth College, Hanover, NH, SAD, 1997. 176
9. Harrison, C., Chess, D., and Kershenbaum, A.: Mobile Agents: Are they a Good Idea? Homepage of IBM T.J. Watson Research Center, 1995. 168
10. Hollingsworth, D.: Workflow Management Coalition - the Workflow Reference Model. The Workflow Management Coalition Specification - Doc. No. TC00-1003, 1995. 168, 169
11. Ivanović, M., Badjonski, M., Budimac, Z.: Programming language PLES and its usage. In Proc. of XLI Conf. of ETRAN (Zlatibor, Yugoslavia), Djordjević, J., Stanković, S., eds., 1997, pp. 40-42 (in Serbian). 173
12. Karnik, N.M., Tripathi, A.: Design Issues in Mobile-Agent Programming Systems. IEEE Concurrency, July-Sept. 1998, pp. 52-61. 168
13. Lingnau, A. and Drobnik, O.: An Infrastructure for Mobile Agents: Requirements and Architecture. Homepage of Jochan Wolfgang Goethe-University, Frankfurt am Main, 1997. 168
14. Marwood, D.: Extending Applications to the Network. M.Sc. Thesis, Dept. of Computer Science, The University of British Columbia, 1998. 168
15. Mitsubishi Electric: Mobile Agent Computing: White Paper. Homepage of Horizon Systems Lab, 1998. 168
16. Morreale, P.: Agents on the Move. IEEE Spectrum, April 1998, pp. 34-41. 168
17. Rothermel, K. And Straßer, M.: A Protocol for Preserving the Exactly-Once Property of Mobile Agents. Bericht 1997/18, Stuttgart University, Faculty of Informatics, 1997. 175
18. Strasser, M., Baumann, J., and Hohl, F.: Mole - A Java Based Mobile Agent System. University of Stuttgart homepage, 1996. 168
19. Sheth, A., Kochut, K., Miller, J., Worah, D., Das, S., Lin, C., Palaniswami, D., Lynch, J. and Shevchenko, I.: Supporting State-wide Immunization Tracking using Multi-Paradigm Workflow Technology, Proc. of 22nd VLDB Conference (Bombay, India), 1996. 176

Pattern-Oriented Hierarchical Clustering*

Tadeusz Morzy, Marek Wojciechowski, Maciej Zakrzewicz

Poznan University of Technology
Institute of Computing Science
ul. Piotrowo 3a, 60-965 Poznan, Poland
Tadeusz.Morzy@put.poznan.pl
Marek.Wojciechowski@cs.put.poznan.pl
Maciej.Zakrzewicz@cs.put.poznan.pl

Abstract. Clustering is a data mining method, which consists in discovering interesting data distributions in very large databases. The applications of clustering cover customer segmentation, catalog design, store layout, stock market segmentation, etc. In this paper, we consider the problem of discovering similarity-based clusters in a large database of event sequences. We introduce a hierarchical algorithm that uses sequential patterns found in the database to efficiently generate both the clustering model and data clusters. The algorithm iteratively merges smaller, similar clusters into bigger ones until the requested number of clusters is reached. In the absence of a well-defined metric space, we propose the similarity measure, which is used in cluster merging. The advantage of the proposed measure is that no additional access to the source database is needed to evaluate the inter-cluster similarities.

1 Introduction

Clustering is one of the most popular data mining methods [2] [3] [4] [5] [6] [7] [8] [9] [11]. It consists in discovering interesting data distributions and patterns in very large databases. Given *k* data points in a *d*-dimensional metric space, the problem of clustering is to partition the data points into *n* clusters such that the data points within a cluster are closer (more similar) to each other than data points in different clusters. Clustering is often used for market segmentation, in which the customers are divided into groups based on the similarity of their characteristics.

Clustering algorithms typically determine *n* partitions that optimize some criterion function. The most commonly used criterion is the square-error criterion defined as follows:

$$E = \sum_{i=1}^{n} \sum_{p \in c_i} \left\| p - m_i \right\|^2 \qquad (1)$$

* This work was partially supported by the grant no. KBN 43-1309 from the State Committee for Scientific Research (KBN), Poland.

Eder et al. (Eds.): ADBIS´99, LNCS 1691, pp.179 -190, 1999.
© Springer-Verlag Berlin Heidelberg 1999

where m_i is the mean of cluster c_i, p is a data point, and E is the square error. Many clustering algorithms employ the idea of *hierarchical clustering*, which consists in merging pairs of similar clusters to form new larger clusters.

Traditional clustering approaches deal with points in d-dimensional space. However, not all types of information can be represented in this form. In many applications, users operate on databases of event sequences, such as customer purchase history given in Figure 1, where an ordered set of purchased products is stored for each customer. Let us consider the general problem of the similarity of sequences. It seems that e.g. the sequences $a \rightarrow b \rightarrow c$ and $b \rightarrow c \rightarrow d$ are similar since the both contain the same subsequence $b \rightarrow c$. But what can we say about the similarity of the sequences e.g. $a \rightarrow b$ and $c \rightarrow d$? We claim that these two sequences also can be considered similar, if there are many other sequences in the database, that contain them both, e.g. $a \rightarrow b \rightarrow c \rightarrow d$, $a \rightarrow c \rightarrow b \rightarrow d$, etc. Therefore, we assume that two sequences are similar if either they contain the identical subsequences, or their subsequences have the tendency to co-occur together in some other sequences. For example, the customer *103* is more similar to the customer *104* than to the customer *105* since the pair *103-104* has a common subsequence (*tv_set* $\rightarrow$ *vcr* $\rightarrow$ *cassette*) while the pair *103-105* has no common subsequences. We are interested in clustering sequences based on such intuitive similarity measure. We notice that this problem cannot be solved using traditional clustering methods because: 1. the sequences are variable-length, 2. the sequences cannot be represented in a d-dimensional metric space, and 3. no natural distance function is available.

In this paper we address and solve the problem of partial clustering of sequential data. We are interested in discovering an arbitrary number of possibly overlapping clusters that hold the customers, whose behavior is similar to each other. We refer to our clustering method as to *partial* clustering, because we allow the customers who are not similar to any other not to be covered by any cluster, and we allow a customer to belong to more than one cluster. To perform the partial clustering of sequential data, we employ the idea of sequential pattern discovery [1] [10]. A sequential pattern is a frequently occurring subsequence of database sequences. Sequential pattern discovery consists in finding all sequential patterns, whose frequency is above some user-defined minimum value. Thus, the discovered sequential patterns represent the most common subsequences within the database sequences and can be used to determine their similarity. For example, the sequential patterns that can be discovered in the database from Figure 1 are: *'tv_set* $\rightarrow$ *vcr* $\rightarrow$ *cassette'* (contained in two database sequences), *'book* $\rightarrow$ *c_disk'* (also contained in two), etc. Each of those sequential patterns says that a number of customers bought one product, later on they bought some other product, and so on.

The presented algorithm uses sequential patterns discovered in the database to generate both the clustering model and cluster contents. For example, our algorithm executed on the database from Fig. 1 with *n=2* gives two clusters based on the following clustering model:

- Cluster 1: described by the patterns: *tv_set* $\rightarrow$ *vcr* $\rightarrow$ *cassette* and *bicycle* $\rightarrow$ *b_ball*; the cluster contains the following customers: 102, 103, 104,
- Cluster 2: described by the patterns: *book* $\rightarrow$ *c_disk* and *lamp* $\rightarrow$ *pillow*; the cluster contains the following customers: 101, 105.

cust_id	Sequence
101	lamp → l_bulb → pillow → book → c_disk
102	t_rocket → bicycle → b_ball → s_bindings
103	tv_set → vcr → c_phone → cassette
104	bicycle → tv_set → b_ball → vcr → cassette
105	book → c_disk → dryer → lamp → pillow → d_washer

Fig. 1. Example of customer purchase history database

1.1 Related Work

In recent years, a number of clustering algorithms for large databases has been proposed. In [8], a clustering method based on randomized search, called CLARANS has been introduced. CLARANS was dedicated to solve problems of data mining in spatial databases. The problem of clustering in large spatial databases was also addressed in [2] and [3]. In [11], the authors presented a clustering method named BIRCH whose I/O complexity was a little more than one scan of the data. In [5], the hierarchical clustering algorithm named CURE was presented. The algorithm was designed for identifying clusters having non-spherical shapes. In [6] and [7], a method for hypergraph-based clustering of data in a high dimensional space has been presented. In [9], a clustering method for data without distance functions was considered, and the proposed algorithm tried to group together records that had frequently co-occurring items. The implementation of the algorithm used frequent item sets discovered by the association rule algorithm as hypergraph edges. [4] described a novel approach for clustering collections of sets, and its application to the analysis and mining of categorical data. The proposed algorithm facilitated a type of similarity measure arising from the co-occurrence of values in the dataset. Unfortunately, none of the works addressed the problem of clustering of sequences of events.

The problem of frequent pattern discovery in sequential data was introduced in [1] and three different algorithms were proposed. In [10], the problem was generalized by adding time constraints and taxonomies.

1.2 Paper Outline

The paper is organized as follows. In Section 2, the basic definitions and the formulation of the problem are given. Section 3 contains the problem decomposition and the description of the algorithm for pattern-oriented clustering. The idea behind our algorithm is illustrated by a detailed example. We conclude with a summary and directions for future work in Section 4.

2 Problem Formulation

2.1 Definitions

Let $L = \{l_1, l_2, ..., l_m\}$ be a set of literals called items. A *sequence* $S = <X_1 X_2 ... X_n>$ is an ordered list of sets of items such that each set of items $X_i \subseteq L$. Let the database D be a set of sequences.

We say that the sequence $S_1 = <Y_1 Y_2 ... Y_m>$ *supports* the sequence $S_2 = <X_1 X_2 ... X_n>$ if there exist integers $i_1 < i_2 < ... < i_n$ such that $X_1 \subseteq Y_{i1}$, $X_2 \subseteq Y_{i2}$, ..., $X_n \subseteq Y_{in}$. We also say that the sequence S_2 is a *subsequence* of the sequence S_1 (denoted by $S_2 \subset S_1$).

A *frequent pattern* is a sequence that is supported by more than a user-defined minimum number of sequences in D. Let P be a set of all frequent patterns in D.

A *cluster* c is an ordered pair $<Q,S>$, where $Q \subseteq P$ and $S \subseteq D$, and S is a set of all database sequences supporting at least one pattern from Q. We call Q a *cluster description*, and S a *cluster content*. We use a dot notation to refer to a cluster description as to $c.Q$ and to a cluster content as to $c.S$.

A *union* c_{ab} of the two clusters c_a and c_b is defined as follows:

$c_{ab} = union(c_a, c_b) = < c_a.Q \cup c_b.Q , c_a.S \cup c_a.S >$

Inter-cluster similarity is a co-occurrence function $f(c_1, c_2)$. In this paper, we use the following co-occurrence function:

$$f(C_1, C_2) = \frac{|C_1.S \cap C_2.S|}{|C_1.S \cup C_2.S|} \quad (2)$$

The above similarity function returns values from the range of $<0;1>$, where the value of *1* means that the clusters are identical while the value of *0* means that the clusters exhibit no similarity at all.

2.2 Problem Statement

Given a database $D = \{s_1, s_2, ..., s_n\}$ of data sequences, and a set $P = \{p_1, p_2, ..., p_m\}$ of frequent patterns in D, the problem is to divide P into a set of clusters, such that $\forall_{i,j,i \neq j} C_i.Q \cap C_j.Q = \emptyset$, and inter-cluster similarity is minimized.

3 Pattern-Oriented Hierarchical Clustering

In this section, we describe a new clustering algorithm POPC for clustering large volumes of sequential data. The algorithm implements the general idea of hierarchical clustering. However, instead of starting with a set of clusters containing one data sequence each, our algorithm uses previously discovered frequent patterns and starts with clusters containing data sequences supporting the same frequent pattern. We assume that a well-known algorithm for frequent pattern discovery is executed before.

3.1 Algorithm POPC

The algorithm for partial clustering based on frequently occurring patterns is decomposed into the following phases:

- *Transformation Phase*, which prepares the database for effective similarity evaluations,
- *Merge Phase*, which iteratively reduces the number of clusters by merging the most similar ones,
- an optional *Description Pruning Phase*, which can be applied to compress the generated clustering model.

3.1.1 Transformation Phase

In this phase, the database is transformed into a pattern-oriented form, which is more suitable for evaluating unions and intersections of cluster contents (used in the subsequent phases). For each frequent pattern we keep an ordered list of data sequences supporting the pattern. Each data sequence is represented by its identifier, e.g. in our example of sequences corresponding to lists of products bought by clients of a supermarket, a sequence could be identified by a unique identifier of a client. Sequences that do not support any frequent pattern are ignored.

Each pattern, together with the list of sequences supporting it, constitutes a cluster whose description is a set that contains the pattern as its only element. The cluster's content is made up of a set of data sequences from the list.

The proposed database representation simplifies evaluation of inter-cluster similarities. There is no need to refer to the original database in subsequent phases of the algorithm. Moreover, the size of the transformed database reduces as clusters are being merged together. When the process is finished, the database contains the result of clustering (descriptions and contents of the discovered clusters).

3.1.2 Merge Phase

Figure 2 presents the Merge Phase of the clustering algorithm. First, the m patterns are mapped into m clusters, forming an initial set of clusters C_1, where each cluster is described by exactly one pattern. In the next step, the similarity function values are evaluated for all possible combinations of clusters. The similarity values are stored in a form of a matrix M_1. Next, the algorithm iteratively merges together pairs of clusters according to their similarity values and cluster contents' sizes. In each iteration k, the two most similar clusters $c_a, c_b \in C_k$ are determined, and replaced by a new cluster $c_{ab} = union(c_a, c_b)$. If there are several pairs of clusters having maximal similarity values, then the two clusters having the smallest contents are merged. The actual merging is done by the function called *cluster*, described in detail in Section 3.1.2.2. When the new cluster is created, the matrix containing similarity values has to be re-evaluated. This operation is performed by means of the function called *simeval*, described in Section 3.1.2.1.

The Merge Phase stops when the number of clusters reaches n or when there is no such pair of clusters $c_a, c_b \in C_k$ whose similarity is greater than 0. The latter condition implies that the algorithm may discover a larger number of clusters than requested by a user. In this case, the number of discovered clusters (as well as the fraction of the

original database covered by them) depends on the number and strength of frequent patterns used for clustering. If the quality of clustering is unsatisfactory, the clustering should be repeated with a higher number of frequent patterns (a set of patterns satisfying a lower frequency threshold).

$C_1 = \{c_i: c_i.Q=\{p_i\}, c_i.S=\{s_j: s_j \in D \wedge s_j \textit{ supports } p_i\}\}$;
$M_1 = \textit{simeval}(C_1, \emptyset)$;
k=1;
while $|C_k| > n$ and exist $c_a, c_b \in C_k$ such that $f(c_a,c_b) > 0$ ***do begin***
  $C_{k+1} = \textit{cluster}(C_k, M_k)$;
  $M_{k+1} = \textit{simeval}(C_{k+1}, M_k)$;
  k++;
end;
Answer $= C_k$;

Fig. 2. Merge Phase

3.1.2.1 Similarity Matrix Evaluation: simeval

Similarity matrix M_l stores the values of the similarity function for all possible pairs of clusters in an *l*-th algorithm iteration. The cell $M_l(x,y)$ represents the similarity value for the clusters c_x and c_y from the cluster set C_l (see example in Figure 3). The function *simeval* computes the values of the similarity matrix M_{l+1}, using both the similarity matrix M_l and the current cluster contents. Notice that in all iterations except the first one, the similarity matrix need not be completely re-computed. Only the similarity values concerning the newly created cluster have to be evaluated. Due to diagonal symmetry of the similarity matrix, for *k* clusters, only $(k^2-k)/2$ similarity function values need to be computed before the first iteration, and only $(k-1)$ in the subsequent ones.

In each iteration, the size of the matrix decreases since two rows and two columns corresponding to the clusters merged to form a new one are removed and only one column and one row are added for a newly created cluster.

-	$f(c_2, c_1)$	$f(c_3, c_1)$
$f(c_1,c_2)$	-	$f(c_3, c_2)$
$f(c_1,c_3)$	$f(c_2, c_3)$	-

$f(c_1,c_2) = f(c_2, c_1)$
$f(c_1,c_3) = f(c_3, c_1)$
$f(c_2, c_3) = f(c_3, c_2)$

Fig. 3. Structure of the similarity matrix for three clusters

3.1.2.2 Cluster Merging: cluster

In each iteration, the number of processed clusters decreases by one. The similarity-based merging is done by the function called *cluster*. The function *cluster* scans the similarity matrix and finds pairs of clusters, such that their similarity is maximal. If there are many pairs of clusters that reach the maximal similarity values, then the function *cluster* selects the one with the smallest size of the union of their contents. Notice that no access to the original database is required to perform this phase of the

algorithm. The function *cluster* takes a set of clusters C_k as one of its parameters and returns a set of clusters C_{k+1} such that $C_{k+1} = (C_k \setminus \{c_a, c_b\}) \cup \{c_{ab}\}$, where $c_a, c_b \in C_k$ are clusters chosen for merging and $c_{ab} = union(c_a, c_b)$.

3.1.3 Description Pruning Phase (optional)

The Merge Phase returns the complete set of the requested clusters. However, the clusters may have descriptions that are not minimal. That is, some patterns included in a description might be redundant in such a way that a set of data sequences supporting the pattern will always be a subset of data sequences supporting some other pattern included in the same description. In the optional Description Pruning Phase, descriptions of all clusters can be tested whether they include a pair of patterns p_a, p_b such that $p_a \subset p_b$. If such a pair is found, p_b is removed from the description. Figure 4 presents the Description Pruning Phase of the algorithm.

```
C = the set of clusters generated in the Merge Phase;
for each c_i ∈ C do
    while exist p_a,p_b ∈ c_i.Q such that p_a ⊂ p_b do
        c_i.Q = c_i.Q \ { p_b };
```

Fig. 4. Description Pruning Phase

Notice that it is also possible to perform pruning of descriptions within the Merge Phase for each newly created cluster.

3.2 Example

Consider a database of customer transactions shown in Figure 5. For each transaction, we keep the transaction's time, items bought in the transaction and a unique customer identifier. Figure 6 shows an alternative representation of the database, where an ordered set of purchased items is given for each customer.

Let us assume that a user wants to cluster customers who follow similar frequent buying patterns into three clusters. Figure 7 shows frequent sequential patterns discovered in the database from Figure 5 (with a support threshold of 25%). The clustering algorithm starts with the Transformation Phase, which results in the initial set of clusters shown in Figure 8.

Customer Id	Transaction Time	Items Bought
1	October 10 1998	10 60
1	December 10 1998	20 30
1	December 15 1998	40
1	February 19 1999	50
2	November 10 1998	40
2	November 21 1998	50
2	December 12 1998	10
2	January 18 1999	20 30 70
3	October 15 1998	40
3	November 29 1998	50
3	December 14 1998	10
3	January 22 1999	80
3	February 11 1999	20 30
4	December 20 1998	10
4	February 4 1999	20
5	February 12 1999	80
6	November 1 1998	10
6	November 22 1998	30 90
7	February 1 1999	20 30
8	October 10 1998	60
8	November 22 1998	100
9	January 12 1999	100
10	January 21 1999	90 100

Fig. 5. Database sorted by Customer ID and Transaction Time

ID	Customer sequence
1	< (10 60) (20 30) (40) (50) >
2	< (40) (50) (10) (20 30 70) >
3	< (40) (50) (10) (80) (20 30) >
4	< (10) (20) >
5	< (80) >
6	< (10) (30 90) >
7	< (20) (30) >
8	< (60) (100) >
9	< (100) >
10	< (90 100) >

Fig. 6. Customer-sequence representation of the database

Patterns with support > 25%	
p_1	< (10) (20 30) >
p_2	< (10) (20) >
p_3	< (10) (30) >
p_4	< (20 30) >
p_5	< (10) >
p_6	< (20) >
p_7	< (30) >
p_8	< (40) (50) >
p_9	< (40) >
p_{10}	< (50) >
p_{11}	< (100) >

Fig. 7. Pattern set used for clustering

Cluster	Description	Sequences
c_a	p_1	1, 2, 3
c_b	p_2	1, 2, 3, 4
c_c	p_3	1, 2, 3, 6
c_d	p_4	1, 2, 3, 7
c_e	p_5	1, 2, 3, 4, 6
c_f	p_6	1, 2, 3, 4, 7
c_g	p_7	1, 2, 3, 6, 7
c_h	p_8	1, 2, 3
c_i	p_9	1, 2, 3
c_j	p_{10}	1, 2, 3
c_k	p_{11}	8, 9, 10

Fig. 8. Pattern-oriented representation of the database

Before the first iteration of the Merge Phase the similarity matrix has to be build. The similarity matrix for the initial set of clusters from Figure 8 is shown in Figure 9.

	c_a	c_b	c_c	c_d	c_e	c_f	c_g	c_h	c_i	c_j	c_k
c_a	x	0.75	0.75	0.75	0.6	0.6	0.6	1	1	1	0
c_b	0.75	x	0.6	0.6	0.8	0.8	0.5	0.75	0.75	0.75	0
c_c	0.75	0.6	x	0.6	0.8	0.5	0.8	0.75	0.75	0.75	0
c_d	0.75	0.6	0.6	x	0.5	0.8	0.8	0.75	0.75	0.75	0
c_e	0.6	0.8	0.8	0.5	x	0.66	0.66	0.6	0.6	0.6	0
c_f	0.6	0.8	0.5	0.8	0.66	x	0.66	0.6	0.6	0.6	0
c_g	0.6	0.5	0.8	0.8	0.66	0.66	x	0.6	0.6	0.6	0
c_h	1	0.75	0.75	0.75	0.6	0.6	0.6	x	1	1	0
c_i	1	0.75	0.75	0.75	0.6	0.6	0.6	1	x	1	0
c_j	1	0.75	0.75	0.75	0.6	0.6	0.6	1	1	x	0
c_k	0	0	0	0	0	0	0	0	0	0	x

Fig. 9. Initial similarity matrix

In the first iteration there are six pairs of clusters having maximal similarity (similarity = *1*): (c_a, c_h), (c_a, c_i), (c_a, c_j), (c_h, c_i), (c_h, c_j) and (c_i, c_j). Since all the six pairs have the same sum of cluster contents' sizes, any of them can be chosen for merging (the actual choice may depend on a particular implementation of the algorithm). In this example we assume that a pair of clusters first found during a scan of the similarity matrix (performed row after row, from left to right) is chosen in such situations. This leads to selecting (c_a, c_h) as the first pair of clusters to be merged. In the next two iterations clusters having similarity = *1* are merged to form c_{ahij}. The database and the similarity matrix after the third iteration are shown in Figure 10. In the fourth iteration, we merge the clusters c_b and c_e (see Figure 11). In the fifth iteration, the clusters c_{be} and c_c are merged to form the intermediate result presented in Figure 12. Then, the merging of the cluster clusters c_d and c_f in the sixth iteration leads to the state illustrated by Figure 13.

Cluster	Description	Sequences
c_{ahij}	p_1, p_8, p_9, p_{10}	1, 2, 3
c_b	p_2	1, 2, 3, 4
c_c	p_3	1, 2, 3, 6
c_d	p_4	1, 2, 3, 7
c_e	p_5	1, 2, 3, 4, 6
c_f	p_6	1, 2, 3, 4, 7
c_g	p_7	1, 2, 3, 6, 7
c_k	p_{11}	8, 9, 10

	c_{ahij}	c_b	c_c	c_d	c_e	c_f	c_g	c_k
c_{ahij}	x	0.75	0.75	0.75	0.6	0.6	0.6	0
c_b	0.75	x	0.6	0.6	0.8	0.8	0.5	0
c_c	0.75	0.6	x	0.6	0.8	0.5	0.8	0
c_d	0.75	0.6	0.6	x	0.5	0.8	0.8	0
c_e	0.6	0.8	0.8	0.5	x	0.66	0.66	0
c_f	0.6	0.8	0.5	0.8	0.66	x	0.66	0
c_g	0.6	0.5	0.8	0.8	0.66	0.66	x	0
c_k	0	0	0	0	0	0	0	x

Fig. 10. Database and similarity matrix after 3 iterations

Cluster	Description	Sequences
c_{ahij}	p_1, p_8, p_9, p_{10}	1, 2, 3
c_{be}	p_2, p_5	1, 2, 3, 4, 6
c_c	p_3	1, 2, 3, 6
c_d	p_4	1, 2, 3, 7
c_f	p_6	1, 2, 3, 4, 7
c_g	p_7	1, 2, 3, 6, 7
c_k	p_{11}	8, 9, 10

	c_{ahij}	c_{be}	c_c	c_d	c_f	c_g	c_k
c_{ahij}	x	0.6	0.75	0.75	0.6	0.6	0
c_{be}	0.6	x	0.8	0.5	0.66	0.66	0
c_c	0.75	0.8	x	0.6	0.5	0.8	0
c_d	0.75	0.5	0.6	x	0.8	0.8	0
c_f	0.6	0.66	0.5	0.8	x	0.66	0
c_g	0.6	0.66	0.8	0.8	0.66	x	0
c_k	0	0	0	0	0	0	x

Fig. 11. Database and similarity matrix after 4 iterations

Cluster	Description	Sequences
c_{ahij}	p_1, p_8, p_9, p_{10}	1, 2, 3
c_{bce}	p_2, p_3, p_5	1, 2, 3, 4, 6
c_d	p_4	1, 2, 3, 7
c_f	p_6	1, 2, 3, 4, 7
c_g	p_7	1, 2, 3, 6, 7
c_k	p_{11}	8, 9, 10

	c_{ahij}	c_{bce}	c_d	c_f	c_g	c_k
c_{ahij}	x	0.6	0.75	0.6	0.6	0
c_{bce}	0.6	x	0.5	0.66	0.66	0
c_d	0.75	0.5	x	0.8	0.8	0
c_f	0.6	0.66	0.8	x	0.66	0
c_g	0.6	0.66	0.8	0.66	x	0
c_k	0	0	0	0	0	x

Fig. 12. Database and similarity matrix after 5 iterations

Cluster	Description	Sequences
c_{ahij}	p_1, p_8, p_9, p_{10}	1, 2, 3
c_{bce}	p_2, p_3, p_5	1, 2, 3, 4, 6
c_{df}	p_4, p_6	1, 2, 3, 4, 7
c_g	p_7	1, 2, 3, 6, 7
c_k	p_{11}	8, 9, 10

	c_{ahij}	c_{bce}	c_{df}	c_g	c_k
c_{ahij}	x	0.6	0.6	0.6	0
c_{bce}	0.6	x	0.66	0.66	0
c_{df}	0.6	0.66	x	0.66	0
c_g	0.6	0.66	0.66	x	0
c_k	0	0	0	0	x

Fig. 13. Database and similarity matrix after 6 iterations

Cluster	Description	Sequences
c_{ahij}	p_1, p_8, p_9, p_{10}	1, 2, 3
c_{bcdef}	p_2, p_3, p_4, p_5, p_6	1, 2, 3, 4, 6, 7
c_g	p_7	1, 2, 3, 6, 7
c_k	p_{11}	8, 9, 10

	c_{ahij}	c_{bcdef}	c_g	c_k
c_{ahij}	x	0.5	0.6	0
c_{bcdef}	0.5	x	0.83	0
c_g	0.6	0.83	x	0
c_k	0	0	0	x

Fig. 14. Database and similarity matrix after 7 iterations

Cluster	Description	Sequences
c_{ahij}	p_1, p_8, p_9, p_{10}	1, 2, 3
c_{bcdefg}	p_2, p_3, p_4, p_6, p_5, p_7	1, 2, 3, 4, 6, 7
c_k	p_{11}	8, 9, 10

	c_{ahij}	c_{bcdefg}	c_k
c_{ahij}	x	0.5	0
c_{bcdefg}	0.5	x	0
c_k	0	0	x

Fig. 15. Database and similarity matrix after 8 iterations

The intermediate results of the seventh and eighth iterations are presented in Figures 14 and 15. After the eighth iteration, the requested number of clusters is reached and the Merge Phase ends. Then, in the optional Description Pruning Phase descriptions of the discovered clusters are being minimized. The following relations between patterns are true: $p_2 \subset p_1$, $p_3 \subset p_1$, $p_4 \subset p_1$, $p_5 \subset p_1$, $p_6 \subset p_1$, $p_7 \subset p_1$, $p_5 \subset p_2$, $p_6 \subset p_2$, $p_5 \subset p_3$, $p_7 \subset p_3$, $p_6 \subset p_4$, $p_7 \subset p_4$, $p_9 \subset p_8$, and $p_{10} \subset p_8$. This leads to removing p_8 from the description of cluster c_{ahij}, and p_2, p_3, and p_4 from the description of cluster c_{bcdefg} because each of them includes some other pattern from the same description, for example $p_9 \subset p_8$ and they are both in the description of cluster c_{ahij}. After completion of the Description Pruning Phase we get the final result of clustering shown in Figure 16.

Cluster	Description	Customer Sequences
c_{ahij}	p_1 = <(10) (20 30)>, p_9 = <(40)>, p_{10} = <(50)>	1, 2, 3
c_{bcdefg}	p_5 = <(10)>, p_6 = <(20)>, p_7 = <(30)>	1, 2, 3, 4, 6, 7
c_k	p_{11} = <(100)>	8, 9, 10

Fig. 16. Discovered clusters

The algorithm found three clusters, two of which overlap (the content of one of them even includes the content of the other, but their descriptions do not imply that). Data sequence of the customer *5* is not contained in any cluster because it did not support any frequent pattern, which is a consequence of the fact that the customer did not follow any typical buying pattern.

4 Conclusions and Future Work

We considered the problem of clustering sequential data in large databases. Due to the limitations of the existing clustering methods, we introduced the new algorithm, which uses frequent patterns to generate both clustering model and cluster contents. The algorithm iteratively merges smaller, similar clusters into bigger ones until the requested number of clusters is reached. In the absence of a well-defined metric space, we propose the cooccurrence-based similarity measure to be used in cluster

merging. The advantage of the proposed measure is that no additional access to the source database is needed to evaluate the inter-cluster similarity.

In the future, we plan to extend this work along the following lines:

- analysis of the methods for disjoint clusters generation, where each source sequence is allowed to belong to one cluster only,
- classification of sequential data, leading to generation of the classification rules.

References

1. Agrawal R., Srikant R.: Mining Sequential Patterns. Proc. of the 11th Int'l Conference on Data Engineering (ICDE), Taipei, Taiwan (1995)
2. Ester M., Kriegel H-P., Sander J., Xu X.: A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. Proc. of the 2nd Int'l Conference on Knowledge Discovery and Data Mining (KDD), Portland, Oregon (1996)
3. Ester M., Kriegel H-P., Xu X.: A Database Interface for Clustering in Large Spatial Databases. Proc. of the 1st Int'l Conference on Knowledge Discovery and Data Mining (KDD), Montreal, Canada (1995)
4. Gibson D., Kleinberg J.M., Raghavan P.: Clustering Categorical Data: An Approach Based on Dynamical Systems. Proc. of the 24th Int'l Conference on Very Large Data Bases (VLDB), New York City, New York (1998)
5. Guha S., Rastogi R., Shim K.: CURE: An Efficient Clustering Algorithm for Large Databases. Proc. of the ACM SIGMOD International Conference on Management of Data, Seattle, Washington, USA (1998)
6. Han E., Karypis G., Kumar V., Mobasher B.: Clustering based on association rules hypergraphs. Proc. Workshop on Research Issues on Data Mining and Knowledge Discovery (1997)
7. Han E., Karypis G., Kumar V., Mobasher B.: Hypergraph Based Clustering in High-Dimensional Data Sets: A summary of Results. Bulletin of the IEEE Computer Society Technical Committee on Data Engineering, Vol.21 No. 1 (1998)
8. Ng R.T., Han J.: Efficient and effective clustering methods for spatial data mining. Proc. of the 20th International Conference on Very Large Data Bases (VLDB) , Santiago de Chile, Chile (1994)
9. Ramkumar G. D., Swami A.: Clustering Data Without Distance Functions. Bulletin of the IEEE Computer Society Technical Committee on Data Engineering, Vol.21 No. 1 (1998)
10. Srikant R., Agrawal R.: Mining Sequential Patterns: Generalizations and Performance Improvements. Proc. of the 5th Int'l Conference on Extending Database Technology (EDBT), Avignon, France (1996)
11. Zhang T., Ramakrishnan R., Livny M.: Birch: An efficient data clustering method for very large databases. Proc. of the ACM SIGMOD International Conference on Management of Data, Montreal, Canada (1996)

Data Mining in a Multidimensional Environment

Holger Günzel, Jens Albrecht, Wolfgang Lehner[1]

Department of Database Systems, University Erlangen-Nuremberg
Martensstr. 3, 91058 Erlangen, Germany
{guenzel,jalbrecht,lehner}@informatik.uni-erlangen.de

Abstract. Data Mining and Data Warehousing are two hot topics in the database research area. Until recently, conventional data mining algorithms were primarily developed for a relational environment. But a data warehouse database is based on a multidimensional model. In our paper we apply this basis for a seamless integration of data mining in the multidimensional model for the example of discovering association rules. Furthermore, we propose this method as a user-guided technique because of the clear structure both of model and data. We present both the theoretical basis and efficient algorithms for data mining in the multidimensional data model. Our approach uses directly the requirements of dimensions, classifications and sparsity of the cube. Additionally we give heuristics for optimizing the search for rules.

1 Introduction

In a dynamic market environment with many competitors it is crucial for an enterprise to have on-line information about its general business figures as well as detailed information on specific topics. Generally, both the research and the commercial area distinguish between two types of information and analysis approaches. On the one hand there is the "query and answer" approach of OLAP (on-line analytical processing, [4]), which requires an intelligent user. On the other hand there is the "automated" hypotheses generation commonly called data mining [6]. After several years of misunderstandings it became obvious that an automated and unguided retrieval of knowledge is impossible. The enormous efforts of the search process and the generation of irrelevant rules and knowledge which are found in a data mining process opened everbodies eyes to the unrealistic illusion. We understand data mining as a method to inductively generate hypotheses about specific data with a search algorithm. Hence, the search of rules, characterizations, cluster or predictions is one step of a knowledge discovery process [5]. These rules have to be interpreted or verified to achieve knowledge.

Our approach combines the world of online analytical processing and data mining. Therefore we call it *multidimensional guided data mining*. The idea is to combine the formal model of multidimensional structures with data mining techniques. OLAP

[1] current address: IBM Almaden Research Center, 650 Harry Road, San Jose, CA 95120, U.S.A.

Eder et al. (Eds.): ADBIS'99, LNCS 1691, pp. 191-204, 1999.
© Springer-Verlag Berlin Heidelberg 1999

requires a consolidated data basis and aggregated data (summary data) for an adequate query response time. Exactly this basis seems appropriate for data mining as well. A data warehouse database [9] is exactly tailored to the users requirements and therefore the ideal starting position for data mining. In our case, we focus on association rules as a data mining technique. Association rules are hypotheses on the correlation between attributes which are often called items. Hypotheses consist always in a specific intensity (confidence) and are only valid on the regarded data.

In the multidimensional view, user queries request data from a data cube stretched by multiple dimensions like product, shop and time. In order to easily specify the result, the dimensions are organized hierarchically, e.g. products can be classified into product families, groups, and areas. This hierarchical structure provides the guideline for the interactive specification and manipulation of queries by OLAP operators like drill-down or slice.

Available algorithms of data mining are not suited for the search for association rules in a multidimensional environment. The structure of the multidimensional model, the classifications on the dimensions and the sparsity of the cube require new techniques. The main approach of this paper considers not the interpretation of association rules but the problems and changes of algorithms for the multidimensional model. [10] distinguishes between intra- or inter-association rules: An example for an intra association rule in the market research area could be: '90 % of costumers who bought a radio, took some batteries as well'. This kind of rules relies only on one dimension. The more important and calculation-intensive rules are inter-association rules like '70 % of the sold radios in Nuremberg were sold in January 1999'.

The structure of the paper is as follows: The next section gives an introduction to conventional data mining algorithms. This demonstrates that our idea is an evolutionary approach. The third section explains the multidimensional model, the problems for data mining, and the visualization of rule generation. The following two sections introduce algorithms searching for itemsets with and without a classification hierarchy. Section 6 sketches our performance study. The paper concludes with a summary.

2 Traditional Data Mining

The problem of mining association rules was first introduced by Agrawal et al. [1]. The search for relationships between attributes with a specific strength was initiated on a flat table with a boolean domain. Therefore, Kamber et al [10] called it intradimensional mining. A set of sales transactions of a supermarket was stored and mined for relations between the sold products. Thus an association rule in general is an expression X => Y, where X and Y are sets of items. An example for a 1-itemset is 'TR-75'; 'TR-75 and Batteries' may be seen as a 2-itemset. The characteristic features of an association rule are the confidence (intensity) and the support, which identify the frequency of the itemsets in the scenario. Therefore to minimize the rules, the user must specify a minimum support and minimum confidence of a rule.

Most of the existing algorithms are derived from Agrawals *apriori* algorithm searching for itemsets which satisfy the required minimum support of the rule [1]. These itemsets are called frequent or large. Apriori means a bottom up search from 1-itemsets to n-itemsets with an estimation of the next step. Instead of controlling or counting all possible combinations of items there is estimated only a smaller set of suitable itemsets. More efficient algorithms (e.g. [2], [14], [15], [19]) reduce the number of passes by partitioning the data, use of samples or additional sets.

After the search for frequent itemsets, the second step generates rules with the required confidence. The confidence originates from the idea of conditional probability, i.e. a division of supports of frequent itemsets. A rule AB => CD requires the itemsets ABCD and AB. Then, the confidence is calculated by support (ABCD) / support (AB). An optional third step discards uninteresting rules with techniques like heuristics, human help or meta rules.

The next generation of algorithms was developed for non-boolean data, i.e. non binary domains. The basic idea is to map the non-boolean data to a boolean data representation [18]. All instances or a class of instances of an attribute build a new attribute. Then between these artificial attributes is mined.

The following step in the evolution of data mining approaches may be seen in multilevel association rules in combination with classification hierarchies. An external classification for raw data is required for searching rules on a generalized level ([7], [8], [3], [17], [16]). The support of the rules increases because of the sum of attribute values and therefore the possibility of a rule existence as well. However, the problem of irrelevant combinations arises in mining with classification hierarchies. Dependencies from classification hierarchies like 'if video than TR-75' become visible. The algorithms differ in the way of finding itemsets. Some start at raw data level and step up the classification [17]. Others start at the top level of the hierarchy and go down to the raw data [7]. The behavior and efficiency always depends on data and classifications and therefore it is hard to estimate the quality.

The further approach is the extension to the multidimensional model. Association rules are found between instances of orthogonal dimensions (interdimensional). Kamber et al. were the first who introduced mining in multidimensional data. But, the presented algorithm in [10] misses the hierarchies which are relevant in the multidimensionality. Our approach extends the basic approaches with a new algorithm for both the model without and with a classification. We place our data mining proposal in the inter-dimensional, non-bool and multilevel location (figure 1 grey area).

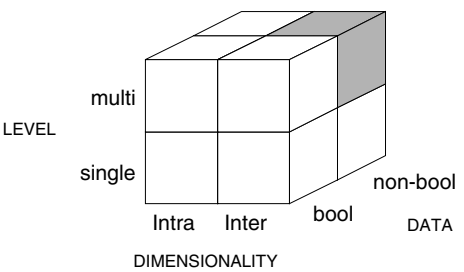


Fig. 1: Several evolutionary steps of data mining

3 Multidimensional Data Mining

Our association rule algorithm is based on the multidimensional model as used in the OLAP application domain. The following description gives a short introduction into the topic and explains the different requirements.

3.1 The Multidimensional Data Model

The following section introduces informally the dimensional and multidimensional structures which build the formal basis of the multidimensional model. The description is divided into the presentation of dimensions (in the one-dimensional context) and multidimensional data cubes (in multidimensional context) both of which are necessary to define the notion of a multidimensional database ([11], [12]). The following examples are always related to figure 2.

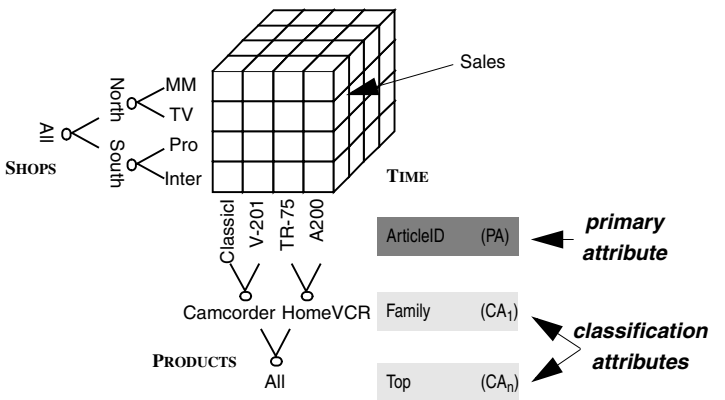


Fig. 2: Sample of a data cube

A *dimension* $\mathcal{D}$ of a multidimensional database is a set of attributes $\mathcal{D} = \{PA\} \cup \{CA_1, ..., CA_p\}$ and a set of functional dependencies between these attributes.

- $\forall i\ (1 \le i \le p)$: $PA \rightarrow CA_i$
  The *primary attribute* PA of a dimension functionally determines all other attributes of a single dimension. The instances of the primary attribute are called *dimensional elements*. *Dimensional elements* (DE) are the basic units of a dimensional structure. They are used to address the micro data, i.e. measures or facts. These single article identifiers reflect the dimensional elements within the product dimension. For example 'TR-75' is a dimensional element for the primary attribute 'ArticleID'.
- $\forall i\ (1 \le i \le p\text{-}1)$: $CA_i \rightarrow CA_{i+1}$
  The *classification attributes* CA_i of a dimension are used to define a multi-level grouping of single dimensional elements and are ordered according to their functional dependencies. The aggregation level or granularity i of a classifica-

tion attribute CA_i is denoted as $[CA_i]$. By definition, the primary attribute has the aggregation level 0 ($PA = CA_0$). Based on the dimensional elements, a balanced tree-structured *classification hierarchy* can be defined to identify business terms like product families, groups, and areas in the product dimension (figure 2). Each *classification node* (C), e.g. 'Camcorder', is an instance of a corresponding *classification attribute* (CA). Thus, on the attribute level, the hierarchy of classification nodes corresponds to a list of classification attributes (CA_i, i=1,...,n) denoted as *categorization* of that dimension. The root node of the classification hierarchy is a specific ALL-node, covering all dimensional elements. For consistency reasons, this root node is the single instance of the highest classification attribute TOP (CA_n), which is always member of each categorization.

Additionally, a multidimensional data cube $\mathcal{M}=(\{\mathcal{D}_1, ..., \mathcal{D}_k\}, \mathcal{S})$ consists of a set of dimensions $\{\mathcal{D}_1, ..., \mathcal{D}_k\}$ and a set of summary attribute S such that $\{\mathcal{D}_1, ..., \mathcal{D}_k\} \rightarrow \mathcal{S}$. An example would be: {Products, Shops, Time}→{Sales}.

3.2 Problems for Data Mining Perspective

The main problems of mining in the multidimensional model are based on the originalities of the model. The dimensional elements, dimensions and classification hierarchies bring along some changes of the requirements and the sequence of the algorithms.

First of all, the dimensional elements define a data cube of multiple dimensions. Since not each shop sells each product at each point in time, it is obvious that this results in an almost empty cube (sparsity). In a common example only 5 % of the cells of a cube contain data. Besides, classifications are not only on a single dimension, but on each dimension. Thus the number of suitable attributes is raised. Additionally, the data volume is much bigger than in conventional databases. Data is collected from several sources in a data warehouse database and is measured in giga- or terabytes. Conventional data mining approaches result in inacceptable performance.

A further arrangement has to be done with summary attributes. The content of the cells influence the usage, i.e. a non summarizable content like price should be treated different than a summarizable. We distinguish between *existential*, *weighted* or *instantial* usage. An existential usage corresponds with the previous utilization in the relational area. Attributes are either existing or not existing. The content of a cell stands for the existence of the combination of several dimensional attributes. Non-summarizable contents like prices must be treated in that way. An instantial usage takes the contents as a new dimensional element or instance of an artificial summary attribute dimension. Rules would be one attribute longer and have a binary content. The most important usage is the weighted approach. The content of a cell supports directly the strength of the rule. Ten sold 'TR-75' in a specific shop in a time interval correspond to ten single sales. An existential usage would only denote the existence of the combination. The instantial way could extent the rules with the item ten. In the following description cells contain summarizable numbers. We only consider the weighted approach, because the others are only special cases.

3.3 Data Mining in the Multidimensional Model

Our formalism for a solution of the problem is chosen similar to Agrawal et al. [1]. Let $I = \{DE_1, DE_2, DE_3...\}$ be a set of items. Items are represented in the multidimensional area as dimensional elements. I_A are only items from dimension A. An itemset X is a set of dimensional elements from different set of items $X \subset (I_A \times I_B \times ...)$. Then, an association rule is an implication of the form $X \Rightarrow Y$ with $X \subset I$, $Y \subset I$ and $X \cap Y = \emptyset$. An itemset is called *large* or *frequent*, if the quantity is bigger than a user defined minimum support (*minsup*). The second user-defined parameter *minconf* defines the minimal confidence of the implication. Confidence is a value characterizing the relation between two itemsets, i.e. when Y is valid, X was already valid.

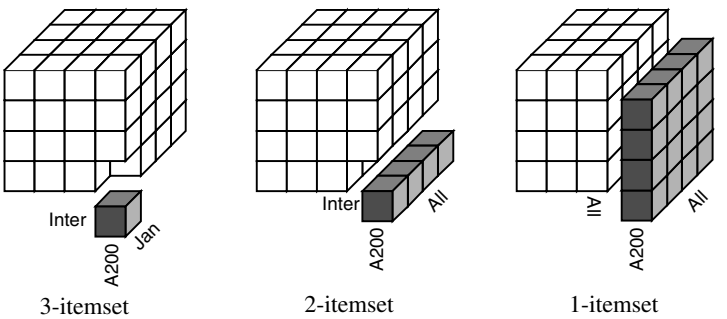


Fig. 3: Visualization of itemsets

In the multidimensional model, it is easy to visualize itemsets (figure 3). There exist only n-itemsets because of the closed environment of the n dimensions. The dimensional attributes 'Inter', 'A200' and 'January' build a 3-itemset (Inter, A200, Jan). The support of a 3-itemset could be read directly from the cell. Then, 'Inter', 'A200' and the complete time dimension build a 2-itemset. In general, all dimensions are fixed with one dimensional attribute. Again, for a (n-1)-itemset a dimension must be left 'open', e.g. all values of a non fixed dimension are summed up in relation to the other fixed attributes. In general, to find a p-itemset from a n-dimensional cube, p dimensions are fixed and (n - p) dimensions are unfixed. Therefore, we can write for a 2-itemset (Inter, A200) or in the equivalent description (Inter, A200, All). In figure 3 the values of the 2-itemset (Inter, A200) the complete time dimension is summed up. That means that a 2-itemset support (Inter, A200) could be found through (Inter, A200, Jan) $\cup$ (Inter, A200, Feb) $\cup$ (Inter, A200, March) $\cup$ (Inter, A200, Apr) or be described as (Inter, A200, All). A 1-itemset (All, A200, All) is found similar to the 2-itemset. In OLAP terminology a 1-itemset is called slice, a 2-itemset a sub-cube.

In the multidimensional model a confidence value is calculated by dividing several subcubes. A confidence of a rule correlates to the division of two supports, whose attributes of the denominator is contained of the numerator. The cube of the numerator is always a sub-cube of the denumerator and totally included. In figure 4 the rule 'if A200 then MM and January' is shown. The confidence is calculated from the support

of the slice (A200) and the 3-itemset (MM, A200, Jan). The maximum quantity of itemsets with n dimensions and m dimensional attributes results in m^n different n-itemsets. Without any restriction, this would result in $m^n * \sum_{i=1}^{m-1} \binom{m}{i}$ rules.

$$\text{conf}_{A200 \Rightarrow MM \wedge Jan} = \frac{\text{support (MM A200 Jan)}}{\text{support (All A200 All)}}$$

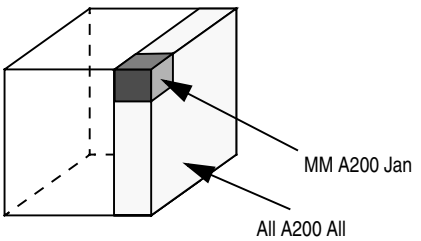


Fig. 4: Visualization of rules

4 Cube-Itemset-Finder

All well-known association rule algorithms do not really fit with the multidimensional model, because they neglect the requirements of the multidimensional model. In this section we propose a new efficient itemset algorithm 'Cube-Itemset-Finder' (CIF) on multidimensional data, which uses several search directions.

4.1 Search Direction

The search for large itemsets is not a question of queries, but for suitable restrictions of the search. The multidimensional data model allows two different ways of searching itemsets: Searching 'bottom-up' from 1-itemset to longer itemsets complies with the apriori approach from Agrawal. A complementary approach (top-down) starts with n-itemsets which are explicitly given in the n-dimensional model and steps down to 1-itemsets.

Bottom-up

The structure of the search for large itemsets could be managed similar to conventional approaches. The threshold value 'minsup' is a decision criterion to generate the k-itemsets from (k-1)-itemsets. It is only worth to count k-itemsets, if (k-1)-itemsets are large. All k-itemsets which base on (k-1)-itemsets have a support lower or equal to the k-itemset, but never higher. A 1-itemset (TR-75) with a support smaller than the minimum support, can not build a 2-itemset like (Inter, TR-75) because the support is smaller or equal. Because of this restriction a lot of irrelevant itemsets are not considered.

Top-down

Another way is to go from n-itemsets down to 1-itemsets. In the multidimensional model, the n-itemsets are given explicitly because of the value of a cell. The support of an n-itemset in the weighted approach is given from the cross product of dimensional attributes. A count of n-itemsets is unnecessary. Therefore, the decision if an itemset is

large could be made directly based on the minsup criterion. If an n-itemset is large, than all sub-itemsets are large, because the supports are rising if the dimensionality is smaller. A large 3-itemset like (Inter, A200, Jan) contains the large itemsets (Inter, A200), (A200, Jan), (Inter, Jan) and (Inter), (A200) and (Jan).

The cells which are not allocated lead to a non-observance of a n-itemsets, because a null value in a cell causes a minimum support smaller than the minsup. A null value in (Inter, A200, Jan) shows that no sub-itemset must be taken, because of irrelevance. But it is possible that other 2-itemsets could be large. These itemsets which are left out are generated if another 3-itemset is large.

Values between null and minsup are problematic. The 3-itemsets (MM,ClassicI,Jan), (TV,ClassicI,Jan), (Pro,ClassicI,Jan) and (Inter,ClassicI,Jan) in figure 5 do not reach the minimum support 3, but the 2-itemset (ClassicI, Jan) has the support 4. In the case that an n-itemset is not large, it is impossible to decide, if an (n-1)-itemset is large. But it is possible to decide if it is worth to compute the (n-1)-itemset. This means, that they are neither *definitely* large (n-1)-itemsets nor it is clear that an (n-1)-itemset is not large. A brute-force approach is a solution generating all sub-itemsets, which could generate too much redundant itemsets. In this case there is no big advantage to the bottom-up approach.

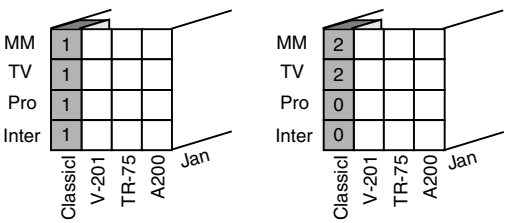


Fig. 5: Problematic 'small' itemsets

A second solution is to skip these non large and not null values. In most of the cases there is no problem because most of these itemsets are found with other large itemsets. The worst case happens, if there is no large itemset, but a lot of small itemsets, which are large in the (n-1)-itemset.

A third approach bases on a heuristics. Therefore, an estimation of the distribution is required. It is obvious, that if a value is close to the minimum support, then the probability is high to reach the minimum support with a the value from a neighbor. Moreover, the probability increases if more cells are involved. A suitable estimation could be done with the function $s * | DE_{n-1} | > minsup$, which involves the distance from the support value of the n-itemset and the number of dimensional elements of the (n-1)-itemset. Using this approach, the cube on the left side of figure 5 would not be searched for the 2-itemset but the cube on the right side. This heuristics means that the search on the (n-1)-level is decided on the n-itemset-level. The justification is based on the generation of the confidence. Rules are only generated if 'extreme' distributions are available and not in homogeneous areas.

4.2 CIF Algorithm

In our proposal, a combination between a top-down and bottom-up approach is given. The algorithm (figure 6) starts with a top-down approach from n-itemsets to 1-itemsets for all large n-itemsets. All n-itemsets with null values are discarded or checked in other large n-itemsets. The problems arise with values between null and large itemsets. Our solution is to build all 1-itemsets for these groups and go up to the (n-1)-itemsets, as long as the minimum support is reached.

Similar to the approach from Agrawal [1] we use different sets to store itemsets during the itemset generation. We need two candidate sets: one for the itemsets which are surely large `C`, and one for the probably frequent itemsets `PF`. The set `L` is directly filled from `C` and contains the itemsets with their support which exceed the minimum support.

In the first step the basic algorithm `CIF_gen-itemset` finds all large n-itemsets (L_n) and generates afterwards all subsets which could be large as well. In PL_n all n-itemsets are stored which do not reach the minimum support but are not null, because their subsets could be large. The sparsity of the cube is therefore not a slow down of the algorithm, because null values are filtered out immediately.

```
 1) procedure CIF_gen-itemset (Cube)
 2) L_n = {all large n-itemsets)
 3) PL_n = {all n-Itemsets with 0 < support < minsup)
 4) foreach l ∈ L_n do
 5)       for (i = n-1; i ≠ 1; i--)
 6)            L_i = subset (L_n+1)
 7) end
 8) foreach p ∈ PF_n do
 9)       L_1 = generate_L_1 (p)
10)       for (k=2; L_k-1 ≠ 0 or k = n-1; k++) do begin
11)       C_k = apriori-gen-mod (L_k-1, p);
12)       L_k = {c ∈ C_k | c ≥ minsup};
13) end
14) return
```

Fig. 6: CIF Algorithm

The first part of the CIF algorithm on page 14 (line 4-7) handles the large itemsets. It generates for all n-itemsets all (n-1)-itemsets, because it is obvious that they are large as well (top-down). The function `subset` generates on the one hand the subsets of the attribute combinations and on the other hand it counts the support of the sub-cube. For performance reasons multiple queries for the same sub-cube have to be skipped.

The second step (bottom-up) between line 8 and 13 remembers conventional approaches in which from 1-itemset successively longer itemsets are generated. But the set is smaller then in the original approaches, because the function `generate_`L_1 only uses elements from PF_n. Afterwards, in `apriori-gen-mod` the combinations of the 1-itemsets are generated. The itemsets in C_k are probably frequent. All frequent

itemsets are stored in the equivalent set L_k, if they reach the minimum support. The loop terminates, if the set C_k is empty and therefore no probably large itemsets are left. At the latest the loop ends on the level n-1, because level n can not be large at all.

5 Classification-Cube-Itemset-Finder (C-CIF)

In comparison to the intra-dimensional approaches, classifications do not effect the subsumption of dimensional attributes from different dimensions, but the relationships of the dimensional attributes of each dimension. Therefore, the support will increase and the number of items keeps constant. If two 1-itemsets (TR75) and (A200) are combined in the classification CA_1 to a 2-itemset (HomeVCR), the support of (A200) and (TR75) has to be added. An example for a 1-itemset with a classification in the product dimension is shown in figure 7.

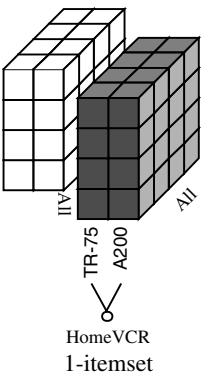


Fig. 7: Visualization of itemsets with classifications

Again there are several possibilities to start searching itemsets with classifications. In general, the same criteria are used. An itemset with an item on a classification CA_n has a support less than or equal to the CA_{n+1}. If the support of (HomeVCR) is less than the minimum support, then the (A200) or (TR75) itemsets never reach the minsup and it is not even worth to count the supports.

The other direction contains similar heuristics. If an itemset on a CA_n level reaches the minsup than on all classification levels CA_{n+1} reach it as well. If the minsup is not reached then we skip the further search. Again, as in the case without classifications the problems result from supports between null and minsup.

5.1 C-CIF Algorithm

Classifications extend the search for frequent itemsets in a cube. The general method of CIF can be used, because a classification is merely an additional disposition of a dimension. The left side of figure 8 shows that there is no difference between the step from a 2-itemset of dimensional elements (Inter, ClassicI) to a 1-itemset (ClassicI) which corresponds to (All, ClassicI) or to a 2-itemset with classifications like (South, ClassicI). It is only a step from a child to a parent node. There exists as well the criterion that a sub-cube is large, if the parent sub-cube is large. But the complexity

increases, because it is possible to combine classification nodes as well, e.g. (South, Camcorder) (figure 8, right side). That means, if a cell of dimensional elements is large, than the sub-cubes on classification level are large as well.

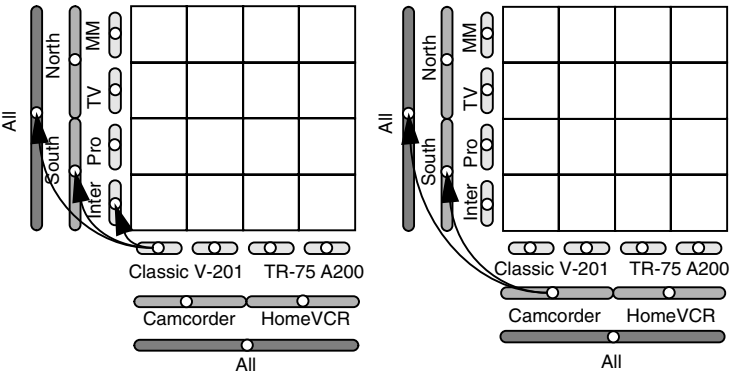


Fig. 8: Classifications and the itemset generation

Our approach needs again several sets for itemsets which are definitely large and some which are probably large. As mentioned, the problematic cases are cells which are neither null and definitely not large and the itemsets which are large. Similar to the CIF algorithm these cases are treated with a 'bottom-up' approach. An extension is required which searches from 1-itemsets not only the n-itemsets, but the classification nodes as well. We say 'top-down' the classification hierarchy. The method stops, if the support is smaller than the minimum support or we reach the (n-1)-itemset.

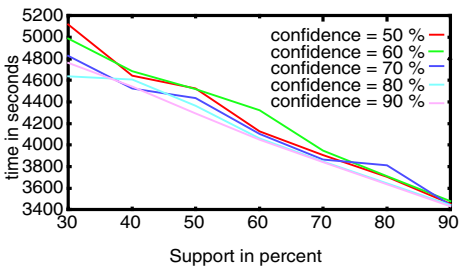
```
1) procedure C-CIF_gen-itemset(Cube)
2) L_n = {all large n-itemsets from dimensional attributes and
   classifications}
3) PL_n = {all n-itemsets from dimensional attributes and
   classifications with 0 < support < minsup}
4) foreach l ∈ L_n do
5)       for (i = n-1; i ≠ 1; i--)
6)            L_i = ext_subset (L_n+1); //with classifications
7) end
8) foreach p ∈ PL_n do
9)       L_1 = generate_L_1 (p);      // Generation of
                                       1-Itemsets on level All
10)      for (k=2; L_k-1 ≠ 0 or k = n-1; k++) do begin
11)      C_k = apriori-gen-mod-clas (L_k-1, p);
12)      L_k = {c ∈ C_k | c ≥ minsup};
13) end
14) return
```

Fig. 9: C-CIF Algorithm

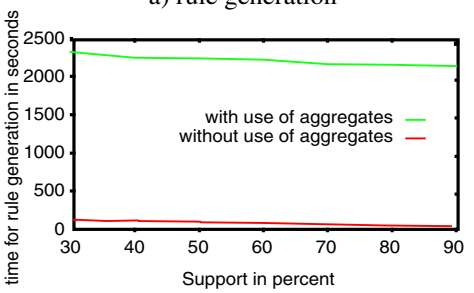
The function `C-CRA_gen-itemset` (figure 9) contains two fundamental changes. Itemsets are searched between dimensional elements and classification nodes. The structure is very similar to the previous deliberations. If a cell is large, then the all father nodes are large as well. If it is not large, the father still could be large. In this case the top-down approach reduces insignificant searching.

The sets L_n, PL_n and C_n have to be extended, in order to contain not only dimensional elements but classification nodes as well. Furthermore the subset generation has to be extended. `Ext_subset` generates not only the subsets of attributes, but also sets of classifications which are probably large. The cells and classification nodes which are not large are treated in the second step with `apriori-gen-mod-clas`.

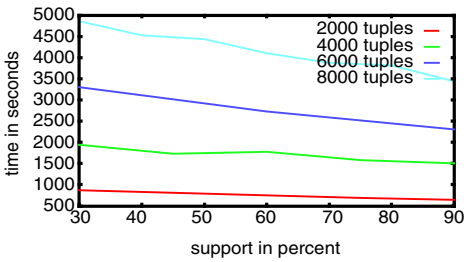
6 Performance Evaluation



a) rule generation



b) use of aggregates



c) database size

Fig. 10: Performance studies

In this section, we evaluate the performance of the CIF algorithm of the previous section. Our data was synthetic and in homogenous distribution. Therefore, the performance could be better in real life application, because of the missing sparsity. We performed our experiments on a Sun E 3000 workstation with 2 CPUs and 512MB of main memory running Sun Solaris 2.6 and JDK 1.1.5. The data of 8056 tupels was stored on an Oracle database system 7.3.2.3 on a DEC Alpha AXP 3000/800 with 187 MB main memory.

We used a so-called relational OLAP (ROLAP) approach instead of a multidimensional approach (MOLAP) storing data in a multidimensional database system. The data were modelled in a *star schema* which contains dimension tables for the hierarchies and a fact table for the dimensional attributes and measures. We started our tests without classifications and without additional database support like indexes or aggregations. Figure 10 (a) shows the correlation between time and support with several minimum confidences. For all confidences, the algorithm needed less time, if the minimum support was increased. Furthermore, the algorithm needs longer than normally required

because we calculate all aggregates for the OLAP scenario as well [13]. Another scenario (figure 10, b) compares the rule generation with and without aggregation calculation. Several minimum supports and a minimum confidence of 70% are shown. The time for rule generation without use of aggregates is more than 250 times higher than without use of aggregates. In figure 10 (c) a test with different data warehouse sizes is presented. The algorithm performs like expected, i.e. the time goes up with the size of the warehouse but less than linear.

7 Conclusion

The basic idea of the paper was to combine data mining and OLAP. Therefore, the idea of data mining in a multidimensional environment is presented. We applied the traditional proposal of dividing the association rule generation in two steps. The first step searches for suitable itemsets and the second generates rules. We proposed one algorithm for searching of large itemsets without classification hierarchies. Our focus was more on the methodology than on evaluation of the usefulness of the rules. With the requirements of the multidimensional model, we have discussed several possibilities. Furthermore we have shown that classifications could be treated with a similar algorithm. The rule generation corresponds to the traditional ideas [1].

A further usage of heuristics or the usage of samples could be useful. The sketched approach for finding large itemsets presents a reliable way of finding all rules. Another interesting approach could be the combination of finding itemsets and rules. Hence, the calculation of rules could be reduced with an estimation of rules by the confidence of existing rules.

References

1 Agrawal, R.; Imielinski, T.; Swami, A.: Mining Association Rules between Sets of Items in Large Databases, in: *Proceedings of the 1993 ACM International Conference on Management of Data* (SIGMOD'93, Washington, D.C., May 26 - 28), 1993, pp. 207-216

2 Agrawal, R.; Mannila, H.; Srikant, R.; Toivonen, H.; Verkamo, A.I.: Fast Discovery of Association Rules, in: [6], pp. 307 - 328

3 Cheung, D. W.; Ng, V.T.; Tam, B.W.: Maintenance of Discovered Knowledge: A Case in Multi-level Association Rules, in: *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining* (KDD'96, Portland, Oregon, Aug. 2 - 4), 1996, pp. 307-310

4 Codd, E.F.; Codd, S.B.; Salley, C.T.: *Providing OLAP (On-Line Analytical Processing) to User Analysts: An IT Mandate*, White Paper, Arbor Software Cooporation, 1993

5 Fayyad, U.M.; Piatetsky-Shapiro, G.; Smyth, P.: From Data Mining to Knowledge Discovery, in [6], pp. 1 - 34

6 Fayyad, U.M.; Piatetsky-Shapiro, G.; Smyth, P.; Uthurusamy, S.: *Advances in Knowledge Discovery and Data Mining*, AAAI/MIT Press, 1996

7 Han, J.; Fu, Y.: Discovery of Multiple-Level Association Rules from Large Databases, in: *Proceedings of the 21st International Conference on Very Large Databases* (VLDB'95, Zurich, Switzerland, Sept. 11 - 15), 1995, pp. 420-431

8 Han, J.: Mining Knowledge at Multiple Concept Levels, in: *Proceedings of the 4th International Conference on Information and Knowledge Management*, (ACM CIKM, Baltimore, Nov. 29 - Dec. 2), 1995, pp. 19-24

9 Inmon, W.H.: *Building the Data Warehouse*, 2. edition. New York, Chichester, Brisbane, Toronto, Singapur: John Wiley & Sons, Inc., 1996

10 Kamber, M.; Han, J.; Chiang, J.Y.: Metarule-Guided Mining of Multi-Dimensional Association Rules Using Data Cubes, in: *Proceeding of the 3rd International Conference on Knowledge Discovery and Data Mining*, (KDD'97, Newport Beach, California, Aug. 14-17), 1997

11 Lehner, W.; Albrecht, J.; Wedekind, H.: Multidimensional Normal Forms, in: *10th International Conference on Scientific and Statistical Data Management* (SSDBM'98, Capri, Italy, July1-3), 1998

12 Lehner, W.: Modeling Large Scale OLAP Scenarios, in: *6th International Conference on Extending Database Technology* (EDBT'98, Valencia, Spain, March 23 - 27), 1998

13 Lehner, W.; Ruf, T.; Teschke, M.: Improving Query Response Time in Scientific Databases using Data Aggregation - A Case Study, in: *7th International Conference and Workshop on Database and Expert Systems Applications* (DEXA'96, Zurich, Switzerland, Sept. 9 -13), 1996

14 Mannila, H.; Toivonen, H.; Verkamo, A.I.: Efficient Algorithms for Discovering Association Rules, in: *Proceedings of the AAAI'94 Workshop on Knowledge Discovery in Databases* (KDD'94, Seattle, WA, July), 1994, pp. 181-192

15 Savasere, A.; Omiecinski, E.; Navathe S.: An Efficient Algorithm for Mining Association Rules in Large Databeses, in: *Proceedings of the 21st International Conference on Very Large Databases* (VLDB'95, Zurich, Switzerland, Sept. 11-15), 1995, pp. 432-444

16 Shen, L.; Shen, H.: Mining Flexible Multiple-Level Association Rules in All Concept Hierarchies, in: *Proceedings of the Ninth International Workshop on Database and Expert Systems Applications* (DEXA 1998, Vienna, Austria, August 24-28), 1998, pp. 786-795

17 Srikant, R.; Agrawal, R.: Mining Generalized Association Rules, in: *Proceedings of the 21st International Conference on Very Large Databases* (VLDB'95, Zurich, Switzerland, Sept. 11 - 15), 1995, pp. 407-419

18 Srikant, R.; Agrawal, R.: Mining Quantitative Association Rules in Large Relational Tables, in: *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data* (SIGMOD'96, Montreal, Quebec, Canada, June 4 -6), 1996, pp. 1 - 12

19 Toivonen, H.: Sampling Large Databases for Association Rules, in: *Proceedings of 22th International Conference on Very Large Data Bases*, (VLDB'96, Mumbai (Bombay), India, Sept. 3 - 6), 1996, pp. 134-145

Arbiter Meta-Learning with Dynamic Selection of Classifiers and its Experimental Investigation

Alexey Tsymbal[1], Seppo Puuronen[1], Vagan Terziyan[2]

[1] University of Jyväskylä, P.O.Box 35, FIN-40351 Jyväskylä, Finland
{alexey, sepi}@jytko.jyu.fi
[2] Kharkov State Technical University of Radioelectronics, 14 Lenin av.,
310166 Kharkov, Ukraine
vagan@kture.cit-ua.net

Abstract. In data mining, the selection of an appropriate classifier to estimate the value of an unknown attribute for a new instance has an essential impact to the quality of the classification result. Recently promising approaches using parallel and distributed computing have been presented. In this paper, we consider an approach that uses classifiers trained on a number of data subsets in parallel as in the arbiter meta-learning technique. We suggest that information is collected during the learning phase about the performance of the included base classifiers and arbiters and that this information is used during the application phase to select the best classifier dynamically. We evaluate our technique and compare it with the simple arbiter meta-learning using selected data sets from the UCI machine learning repository. The comparison results show that our dynamic meta-learning technique outperforms the arbiter meta-learning significantly in some cases but further profound analysis is needed to draw general conclusions.

1 Introduction

Currently electronic data repositories are growing quickly and contain huge amount of data from commercial, scientific, and other domain areas. The capabilities for collecting and storing all kinds of data totally exceed the development in abilities to analyze, summarize, and extract knowledge from this data. Data mining is the process of finding previously unknown and potentially interesting patterns and relations in large databases [5].

A typical data mining task is to predict an unknown value of some attribute of a new instance when the values of the other attributes of the new instance are known and a collection of instances with known values of all the attributes is given. The collection of the instances with the known attribute values is treated as a training set for a learning algorithm that derives a logical expression, a concept description, or a

Eder et al. (Eds.): ADBIS´99, LNCS 1691, pp.205 -217, 1999.
© Springer-Verlag Berlin Heidelberg 1999

classifier, that is then used to predict the unknown value of the attribute for the new instance [2].

Recently, several parallel and distributed computing approaches have been proposed. The main aim behind these approaches is to search techniques that are suitable for huge amounts of data that cannot efficiently be handled by main-memory-based learning algorithms. It has been shown in [2-4] that parallel and distributed processing provides the best hope of dealing with large amounts of data.

In this paper, we consider an approach that uses classifiers learned on a number of data subsets in parallel and that selects for each new instance the best classifier dynamically. This approach reduces and limits the amount of data inspected by every single learning process and provides dynamic classifier selection, increasing the classification speed without significant losses in the classification accuracy or even with an improvement of the accuracy. The approach is based on the arbiter meta-learning technique [2]. We discuss this technique in chapter 2. In chapter 3, we consider our technique for the dynamic selection of classifiers. In chapter 4, we propose a combination of our dynamic classifier selection technique with the arbiter meta-learning. In chapter 5, we present results of our experiments with the approach, and chapter 6 concludes with a brief summary and further research topics.

2 Arbiter Meta-Learning Technique

In [2-4] the arbiter meta-learning technique was proposed for the parallel integration of multiple classifiers. *Meta-learning* encompasses the use of learning algorithms to learn how to integrate results from multiple learning systems. The approach includes data reduction as a solution to the scaling problem. The whole data set is partitioned into smaller subsets, and learning algorithms are applied on these subsets. This is followed by a part of the learning phase, which combines the learned results. It has been proposed to speed up the process by running the learning programs in parallel using multiple processors. It was shown in experiments that the accuracy would not suffer in such a scheme, as one may presume, in comparison with learning from the entire data set [2].

This meta-learning technique is independent of the underlying learning algorithms employed. Furthermore, it does not require a platform that is especially constructed for parallel processing. Thus, the meta-learning approach is intended to be scalable as well as portable and extensible [2]. In this chapter we discuss both the one-level and multi-level arbiter meta-learning briefly.

2.1 One-Level Arbiter Meta-Learning

An *arbiter* is a classifier that is trained to resolve disagreements between the base classifiers. An arbiter is generated using the same learning algorithm that is used to train the base classifiers. In the arbiter technique, the training set for the arbiter is a subset of the union of the training sets for the base classifiers for which the arbiter is

formed. The arbiter training instances are selected to form a particular distribution of the union set [2]. One scheme to select the instances to the arbiter training set, i.e. the *selection rule,* is to pick up the instances for which none of the classes gathers a majority classification [4]. We shall use this selection rule in our experiments in the form presented in [2]. Thus, the predictions of the learned base classifiers determine the subset of the training set that constitutes the arbiter's training set in the learning phase. When a new instance is classified in the application phase, first the base classifiers and the arbiter generate their predictions. Then an *arbitration rule* generates the final prediction using the predictions made by the base classifiers and the arbiter. The generation of an arbiter has much in common with the boosting technique [12] that also filters training instances to train the base classifiers. The approach can be considered as a particular case of the stacked generalization framework [18] that integrates the results of the base classifiers by a trained meta-level classifier.

One arbitration rule that is used to derive the final classification is the following: return the prediction with the majority of occurrences given by the base classifiers and the arbiter. It was used in experiments in [2], and we shall also use it in our experiments in this paper. Preference will be given to the arbiter's choice in the case of a tie. This arbitration rule is based on the most widely used voting principle.

2.2 Arbiter Tree

The one-level arbiter meta-learning is not always able to achieve the same level of accuracy as a global classifier. In [2-4] a hierarchical (multi-level) meta-learning method called *arbiter tree* was considered.

An *arbiter tree* is a hierarchical structure composed of arbiters that are computed in a bottom-up, binary-tree way and it can be generalized to arbiter trees of higher orders. Let there be k base level classifiers. The lowest level arbiters are initially trained using the outputs of a pair of the base classifiers. Thus $k/2$ arbiters are generated at the lowest level. At the second level, arbiters are trained using the outputs of the lowest level arbiters, and recursively at the higher levels of arbiters. For k subsets and k base classifiers there are $log_2(k)$ levels of arbiters generated [2].

When a new instance is classified by the arbiter tree in the application phase, predictions flow from the leaves to the root of the tree. First, each of the leaf classifiers, i.e. base classifiers, produces its classification of the new instance. From a pair of predictions and the classification of the parent arbiter, a classification is produced as a result of the arbitration rule. This process is applied at each higher level until a final classification is produced at the root of the arbiter tree.

It is noted in [2] that in a distributed environment the union sets need not be formed at one processing site. Rather, one can classify each subset by transmitting each learned classifier to each site, which is used to scan the local data set labeled with the predictions of the classifier. Each classifier is a computational object that is far smaller in size than the training set from which it is derived. To reduce the complexity of learning the arbiter trees, the size of the training sets for the arbiters can

be purposely restricted to be no larger than the training sets used to compute the base classifiers. Thus, the parallel processing time at each level of the arbiter tree is approximately equal throughout the tree. In several experiments in [2], it was shown that the classification accuracy does not suffer too much from such a restriction. Without the restriction the size of the training set for an arbiter would be comparable to the size of the entire training set at the root level. Experiments in [2] showed in some cases an accuracy improvement of this multi-level arbiter tree approach over the one-level techniques, which generally could not maintain the accuracy of the global classifier trained on the whole data set.

3 Dynamic Selection of Classifiers

In [9, 10] we proposed a technique for the dynamic integration of classifiers. In [14,15,17] we considered some medical applications of this technique. This technique is based on the assumption that each base classifier gives the best prediction inside certain subdomains of the whole application domain, i.e. inside its competence areas. The main problem in the technique is to estimate the competence areas of the base classifiers in a way that helps the dynamic selection of classifiers for each new instance. Our goal is to use each base classifier just in the subdomain where it is the most reliable one and thus to achieve overall results that can be considerably better than those achieved using the best individual classifier alone. In this chapter, we describe our dynamic selection technique briefly.

The proposed meta-classification framework consists of two levels. The first level contains base classifiers, while the second level contains a combining algorithm that predicts the error for each of the base classifiers. In the training phase we derive the information about the performance of the base classifiers calculating for each training instance the classification errors. These errors can be binary (i.e. a classifier gives a correct/incorrect classification) or they can be numerical values representing corresponding misclassification costs. This information about the base classifier performance is then stacked (as in stacked generalization [18]) and is later used together with the initial training set as meta-level knowledge to estimate the classification error for a new instance.

The information about the base classifier performance is derived using the cross-validation technique [1,7] for learned base classifiers and directly calculated for heuristic non-learned classifiers.

In [9,10] we considered the weighted nearest neighbor classification (WNN) as the meta-level classifier. The WNN simplifies the training phase of the composite classifier because with the WNN there is no need to train referees, only the base classifier performance matrix is needed. In the application phase, the nearest neighbors of a new instance are found out among the training instances and the performances of the corresponding base classifiers are used to predict the performance of each base classifier. In this calculation, we sum up the corresponding performance values of each classifier using weights that depend on the distances between a new instance and its nearest neighbors in the training set.

The use of WNN as the meta-level classifier is based on the assumption that each classifier has certain subdomains in the space of instance attributes, where it is more reliable than the other classifiers. This assumption is supported by the experiences that base classifiers usually work well not only in certain points of the domain space, but in certain subareas of the domain space. The performance of a classifier usually changes gradually from one instance to another near-by instance. Thus if a classifier does not work well with the instances near the new instance, then it is quite probable that it will not work well with the new instance.

4 Application of Dynamic Integration of Classifiers with Arbiter Meta-Learning

In chapter 2, we discussed the arbiter meta-learning technique, which uses a kind of voting, *arbitration rule*, to integrate multiple classifiers both in the one-level approach and in the arbiter tree approach. The voting technique, however, has several shortcomings (see for example [13]). From our point of view the most important shortcoming is that the voting technique is unable to take into account the local expertise. When a new instance is difficult to classify, then the average classifier will give a wrong prediction, and the majority vote will more probably result in a wrong prediction. In that case, one can advise the use of another arbitration rule as in [4]: if the majority of the base classifiers disagrees with the arbiter, then use the arbiter's prediction. However, the arbiter can itself make mistakes in a situation when the majority of the base classifiers does not make them.

In order to improve the meta-learning, we propose the use of our dynamic classifier selection (chapter 3) as the arbitration rule. This helps the selection of the base classifiers and arbiters so that they will be used in the subarea of their expertise. In this chapter, we consider both one-level and multi-level arbiter meta-learning combined with our dynamic classifier selection, and a general algorithm for the arbiter meta-learning technique with the dynamic classifier selection.

4.1 One-Level Arbiter Meta-Learning with Dynamic Classifier Selection

We suggest that the training phase remains bottom-up and the base classifiers are trained first. Then, an arbiter is formed using the same procedure (selection rule), as in the simple arbiter meta-learning [4]. Next, the meta-level information about the performances of the base classifiers and the arbiter is derived comparing their predictions with the actual classifications included in the training set.

Our application phase is top-down. First, the performances of the base classifiers and the arbiter are estimated for a new instance to be classified. Then, a base classifier or the arbiter with the best estimated performance is launched to make the final classification. Thus, during the application phase only one classifier is launched (not all as in the simple arbiter meta-learning).

4.2 Arbiter Tree with Dynamic Classifier Selection

The first level arbiters are initially trained using the outputs of the pairs of the base classifiers, and, recursively, the arbiters at the higher levels are trained using the outputs of the subtrees. An example of an arbiter tree that uses the dynamic selection of classifiers is shown in Figure 1.

The training phase is similar to the simple arbiter tree (chapter 2) with added collection of the classification error information. Let there be initially four training subsets T_1 - T_4. First, the four classifiers C_1 - C_4 are generated in parallel for each T_1 - T_4. Then the union of T_1 and T_2 is classified by C_1 and C_2, and these predictions are used to form the training set for the arbiter A_{12} using the selection rule. Next, the arbiter is trained using the same learning algorithm, and it is used to classify the instances in the union of T_1 and T_2. Based on the classification results of C_1, C_2, and A_{12} the error information of these classifiers on the union of the training sets T_1 and T_2 is collected for dynamic selection DS_{12}. The subtree rooted with DS_{34} is trained using the same procedure as for DS_{12} using the union of T_3 and T_4. After that the union of T_1, T_2, T_3, and T_4 is classified by the subtrees rooted with DS_{12} and DS_{34}. The root arbiter $A_{12,34}$ is formed and the dynamic selection classifier $DS_{12,34}$ is generated, completing the arbiter tree.

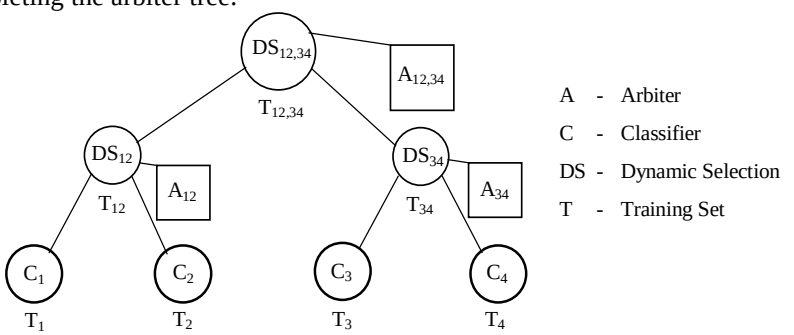


Fig. 1. A simple arbiter tree with the dynamic selection of classifiers

The application phase, however, is completely different from the application phase of the simple arbiter tree technique. In our approach, this phase is a top-down procedure using the attribute values of a new instance as input. First, for a new instance the topmost dynamic selection classifier $DS_{12,34}$ estimates the performances of the subtrees DS_{12}, DS_{34}, and the arbiter $A_{12,34}$. The subtree or the arbiter with the best performance estimate is selected. If the root arbiter $A_{12,34}$ is selected, it is used to make the final classification. Otherwise, the dynamic selection classifier of the selected subtree (either DS_{12} or DS_{34}) is used to estimate the performances of the lower level classifiers and the arbiter. One of the classifiers or the arbiter with the best performance is selected to make the final classification. Thus, during the application phase, at worst as many classifiers are launched as there are levels in the tree. In the simple arbiter tree approach all the classifiers and arbiters must be launched during this phase.

```
Node current node at the tree being learned
fs the first subset for the current node
nos number of subsets for the current node
arity the arity of the tree being learned
Subtree_1...Subtree_arity subtrees or base classifiers
  of the current node
TS_i i-th training subset for meta-learning
C_i base classifier trained on the i-th subset
Arbiter arbiter classifier at the current node
Instance a new instance to be classified
Best_Classifier a subtree, classifier or arbiter
  selected according to the local performance info

Procedure Dynamic_Arbiter_Meta-Learning(Node, fs, nos)
Begin

   If nos=arity
   then
      {train base classifiers}
      Subtree_1=Train(C_fs);
                . . .
      Subtree_arity=Train(C_fs+arity-1);
   else
      {develop the tree further}
      Dynamic_Arbiter_Meta-Learning(Subtree_1,
      fs, nos/arity);
                . . .
      Dynamic_Arbiter_Meta-Learning(Subtree_arity,
      fs+(arity-1)*(nos/arity), nos/arity);
   endif
   {Derive the performance info for
   Subtree_1...Subtree_arity on TS_fs,...,TS_fs+nos-1}
   Evaluate(Subtree_1, fs, nos)
                . . .
   Evaluate(Subtree_arity, fs, nos)
   {Generate a set for the arbiter using
   a selection rule}
   Selection_Rule(fs, nos);
   {train the arbiter}
   Train(Arbiter);
   {Derive the performance info for Arbiter
   on TS_fs,...,TS_fs+nos-1}
   Evaluate(Arbiter, fs, nos);

End
```

```
Function Dynamic_Arbiter_Tree_Application(Node,
   Instance) returns class of Instance
Begin

   If Node is a base classifier or arbiter
   then return Node.Classify(Instance)
   else
      {Select the best classifier using the performance
      info in near-by instances to Instance}
      Best_Classifier=Select_Classifier(Subtree1, ...,
      Subtreearity, Arbiter)
      return Dynamic_Arbiter_Tree_Application
             (Best_Classifier, Instance);
   endif

End
```

Fig. 2. A general algorithm of the arbiter meta-learning with the dynamic classifier selection

Two recursive algorithms for the learning and application phases of the arbiter meta-learning with the dynamic classifier selection are presented in Figure 2. The procedure *Dynamic_Arbiter_Meta-Learning* implements the learning phase of the technique, while the *Dynamic_Arbiter_Tree_Application* implements the application phase. The goal of the learning phase is to build an arbiter tree recursively and to collect the performance information at each node. The goal of the application phase is to use the performance information at each node to select an appropriate classifier for the final classification. These procedures can be applied for the generation and use of trees with different arity and different number of subsets. For example, to train a binary arbiter tree on 32 subsets it is necessary to set *arity=2* and to execute *Dynamic_Arbiter_Meta-Learning*(*Root*, 1, 32), where *Root* denotes the root node of the arbiter tree being learned. When an *Instance* is classified, *Dynamic_Arbiter_Tree_Application*(*Root*, *Instance*) should be executed.

5 Experimental Evaluation

In this chapter, we present an experimental evaluation of the arbiter meta-learning with the dynamic classifier selection. We compare it with the simple arbiter meta-learning. First we present the experimental setting and then describe the data sets and the results of our experiments.

In our experiments, we use the common technique used in evaluating the accuracy of a learning algorithm, the cross-validation technique [1,7]. In the cross-validation technique, all the instances of the training set are divided randomly into *v* approximately equal sized partitions that are usually called *folds*. Each classifier being evaluated is then trained on *v-1* folds *v* times and tested *v* times on the fold

being held-out. We use 10-fold cross-validation, and the averages of 10-fold cross-validation runs are presented in Figure 3. We measure statistical significance of the difference of averages by using the paired differences *t*-test based on train/test cross-validation splits.

In our experiments, we use the C4.5 inductive learning algorithm taken from the machine learning library in C++ (MLC++)[6]. The experiments are implemented within the MLC++ framework. In all the experiments, we use only the Euclidean distance metrics (the standard squared-distance metrics) for finding the nearest neighbors. We vary the number of equi-sized subsets of the training data from 2 to 32 ensuring that the subsets are disjoint with the same distribution of instances of each class. Stratified sampling was made so that each training subset represents a good but smaller model of the entire training set. No pairing strategy for the tree generation and no restriction on the size of the data set for training an arbiter is used, because the main goal of the experiments is to compare our technique with the simple arbiter meta-learning. The experiments with one-level classification trees (*n*-ary trees on *n* training subsets) and with binary multi-level classification trees are conducted on four datasets from the UCI machine learning repository: three MONK's problem datasets donated by Sebastian Thrun and the Tic-Tac-Toe Endgame dataset donated by David W. Aha [8].

The MONK's problems are a collection of three artificial binary classification problems over the same six-attribute discrete domain (a1,…,a6). All MONK's datasets contain 432 instances without missing values, representing the full truth tables in the space of the attributes. The "true" concepts MONK-1, MONK-2, and MONK-3 underlying each MONK's problem are given by: (a1=a2)or(a5=1) for MONK-1, exactly two of {a1=1, a2=1, a3=1, a4=1, a5=1, a6=1} for MONK-2, and (a5=3 and a4=1)or(a5<>4 and a2<>3) for MONK-3. MONK-3 has 5% additional noise (misclassifications) in the training set. The MONK's problems were the basis of the first international comparison of learning algorithms [16].

The Tic-Tac-Toe Endgame dataset encodes the complete set of possible board configurations at the end of tic-tac-toe games, where "x" is assumed to have played first. The target concept is "win for x" (i.e., true when "x" has one of 8 possible ways to create a "three-in-a-row"). The dataset contains 958 instances without missing values, each with 9 attributes, corresponding to tic-tac-toe squares and taking on 1 of 3 possible values: "x", "o", and "empty".

The experimental results are presented in Figure 3. The four meta-learning techniques described above are analyzed: the one-level simple arbiter meta-learning (*Arbiter*), the one-level arbiter meta-learning with the dynamic classifier selection (*Dynamic*), the simple binary arbiter tree (*Arbiter Tree*), and the binary arbiter tree with the dynamic classifier selection (*Dynamic Tree*). The plotted accuracies in the left charts are the averages of 10-fold cross-validation runs. The accuracy of the global classifier is plotted as "the number of subsets=1" that means that the learning algorithm was applied to the whole training set to produce the baseline accuracy results for comparisons. The charts on the right present the learning curves for the datasets. Each point of a learning curve is an average over 70 runs with random training subsets of appropriate size.

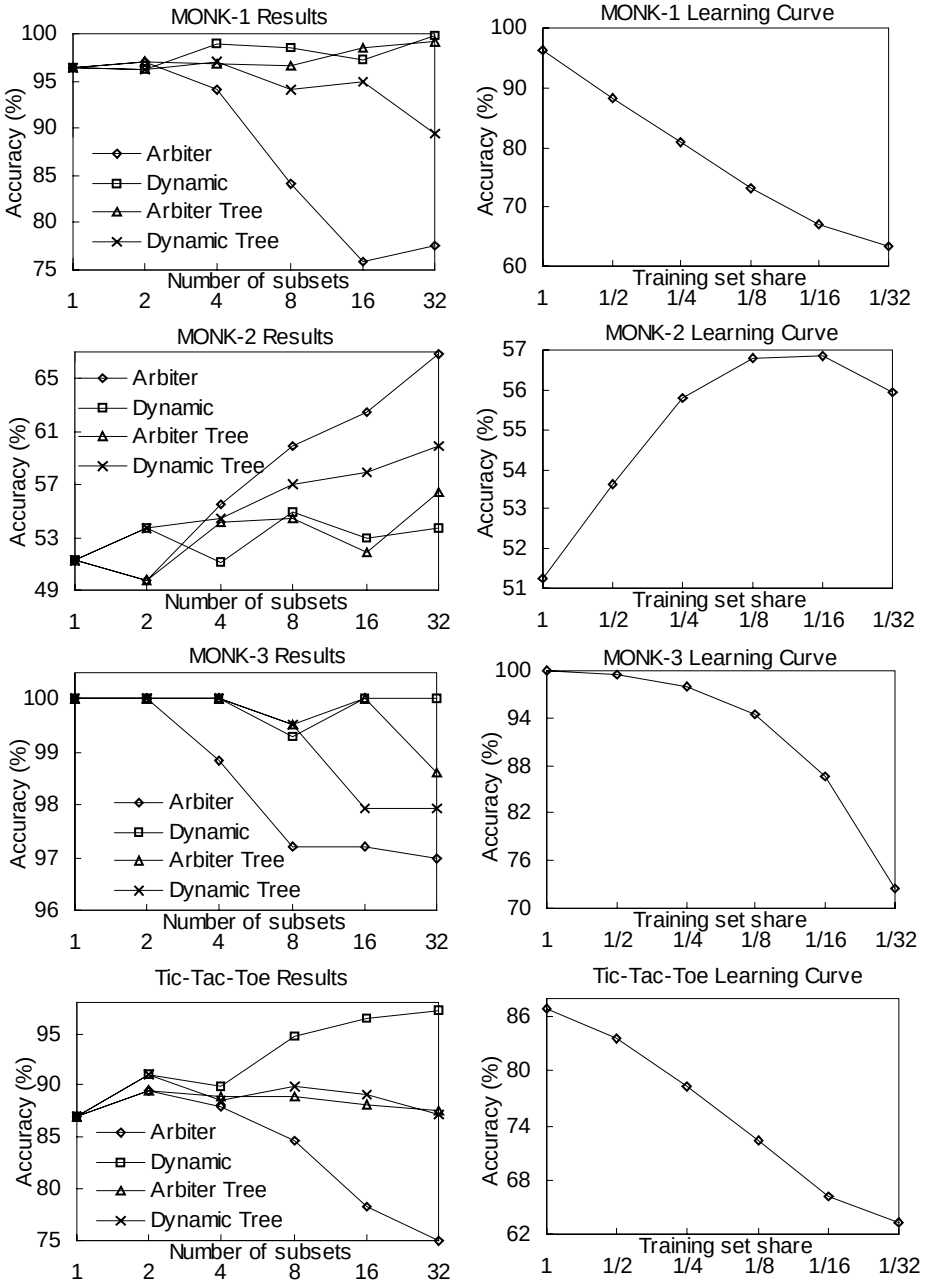


Fig. 3. Experimental results on the 4 data sets from the UCI machine learning repository

Our experimental results support the findings and conclusions made in [4]. All the meta-learning strategies do show a consistent improvement in the classification accuracy over the base classifiers trained on a subset of the training data. This can be seen comparing the resulting charts with corresponding learning curves. Our experimental results show also that both the one-level meta-learning schemes (*Dynamic* and *Arbiter*) and the hierarchical meta-learning schemes (*Dynamic Tree* and *Arbiter Tree*) are often able to sustain the same level of accuracy as a global classifier trained on the entire data set. Thus meta-learning over data partitions can maintain or even boost the accuracy of a single global classifier under certain circumstances. For example, it was done by *Dynamic* on the Tic-Tac-Toe dataset, where the best base classifier on 32 subsets has 73% accuracy, and the global classifier 87% only, but the one-level dynamic arbiter meta-learning classifier has 97% accuracy! It can be seen from the experimental results that this is a very common result. Our experimental results confirm that maximal parallelism can be effectively exploited by the meta-learning over disjoint data partitions without a substantial loss of accuracy. Hierarchically learned classifiers can work better than a single layered meta-learning under certain circumstances. For example, on the MONK-1, MONK-3, and Tic-Tac-Toe datasets the *Arbiter Tree* works significantly better than the *Arbiter*, and on the MONK-2 dataset the *Dynamic Tree* works significantly better than the *Dynamic*.

One can see from the experimental results that under some circumstances our dynamic meta-learning techniques are better than the corresponding simple meta-learning techniques. For example, on the MONK-1, MONK-3, and Tic-Tac-Toe datasets the *Dynamic* is significantly better than the *Arbiter*, and on the MONK-2 dataset the *Dynamic Tree* is significantly better than the *Arbiter Tree*. However, it is not clear under what circumstances our dynamic meta-learning techniques will be better and further studies are needed to gain understanding of these circumstances.

6 Conclusion

We considered the use of the dynamic classifier selection in the arbiter meta-learning. We evaluated our technique and compared it with the simple arbiter meta-learning using selected data sets from the UCI machine learning repository. We showed that our technique often results in the better classification accuracy than the simple arbiter meta-learning. However, multiple experiments on large real-world databases are needed to compare these meta-learning techniques and to find out the conditions under which our technique is better.

There are also many open questions related to our dynamic meta-learning technique which require further experiments. In the above arbiter tree algorithm, the training set grows at each level of the tree, and at the root level a dynamic selection classifier is trained using the whole training set. Unfortunately, such an algorithm is computationally expensive. In [4] it was shown in experiments that the overall accuracy would not suffer significantly if the size of training sets is restricted to the size

of the initial subsets at all nodes. However, this is still open for our approach and it should be checked with multiple experiments.

Another interesting question is "How to pair classifiers in order to construct the best tree?". In [4] Chan proposed three strategies of pairing classifiers: a random pairing, and pairings with the maximal and with the minimal possible arbiter training sets. The experiments in [4] showed that the pairing with the maximal arbiter training sets usually gives better results. We expect that this pairing will also be better in our approach resulting in more diverse arbitrated classifiers. In [10] we have shown that the bigger base classifier diversity results usually in the better coverage by the composite classifier and, consequently, the higher accuracy. The above pairing will generate more compact trees but requires significantly more computer time than the random pairing.

Another open question is "What is the optimal order of the arbiter tree with the dynamic classifier selection?". Experiments in [4] showed that the higher order trees are generally less accurate than the lower order ones. It can be explained by the fact that in the higher order trees it is more difficult for arbiters to arbitrate among bigger number of classifiers. However, this question is also still open for our approach and needs further research.

Acknowledgments: This research is partly supported by the Academy of Finland and the COMAS Graduate School of the University of Jyväskylä. Alexey Tsymbal is thankful to the Kharkov State Technical University of Radioelectronics that allowed him to work in Jyväskylä during the preparation of this article. We would like to thank the UCI machine learning repository of databases, domain theories and data generators for the data sets and the machine learning library in C++ for the source code used in this study.

References

1. Aivazyan, S.A.: Applied Statistics: Classification and Dimension Reduction. Finance and Statistics, Moscow (1989)
2. Chan, P., Stolfo, S.: On the Accuracy of Meta-Learning for Scalable Data Mining. Intelligent Information Systems, Vol. 8 (1997) 5-28
3. Chan, P., Stolfo, S.: Toward Parallel and Distributed Learning by Meta-Learning. In: Working Notes AAAI Work. Knowledge Discovery in Databases (1993) 227-240
4. Chan, P.: An Extensible Meta-Learning Approach for Scalable and Accurate Inductive Learning. PhD Thesis, Columbia University (1996)
5. Fayyad, U., Piatetsky-Shapiro, G., Smyth, P., Uthurusamy, R.: Advances in Knowledge Discovery and Data Mining. AAAI/ MIT Press (1997)
6. Kohavi, R., Sommerfield, D., Dougherty, J.: Data Mining Using MLC++: A Machine Learning Library in C++. Tools with Artificial Intelligence, IEEE CS Press (1996) 234-245
7. Kohavi, R.: A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection. In: Proceedings of IJCAI'95 (1995)

8. Merz, C.J., Murphy, P.M.: UCI Repository of Machine Learning Databases [http://www.ics.uci.edu/ ~mlearn/ MLRepository.html]. Department of Information and Computer Science, University of California, Irvine, CA (1998)
9. Puuronen, S., Terziyan, V., Katasonov, A., Tsymbal, A.: Dynamic Integration of Multiple Data Mining Techniques in a Knowledge Discovery Management System. In: Dasarathy, B.V. (Ed.): Data Mining and Knowledge Discovery: Theory, Tools, and Techniques. Proceedings of SPIE, Vol. 3695. SPIE-The International Society for Optical Engineering, USA (1999) 128-139
10. Puuronen, S., Terziyan, V., Tsymbal, A.: A Dynamic Integration Algorithm with an Ensemble of Classifiers. In: Proceedings ISMIS'99 - The Eleventh International Symposium on Methodologies for Intelligent Systems, Warsaw, Poland, June (1999) (to appear)
11. Quinlan, J.R.: C4.5 Programs for Machine Learning. Morgan Kaufmann, San Mateo, CA (1993)
12. Schapire, R.E.: Using Output Codes to Boost Multiclass Learning Problems. In: Machine Learning: Proceedings of the Fourteenth International Conference (1997) 313-321
13. Skalak, D.B.: Combining Nearest Neighbor Classifiers. Ph.D. Thesis, Dept. of Computer Science, University of Massachusetts, Amherst, MA (1997)
14. Terziyan, V., Tsymbal, A., Puuronen, S.: The Decision Support System for Telemedicine Based on Multiple Expertise. Int. J. of Medical Informatics, Vol. 49, No. 2 (1998) 217-229
15. Terziyan, V., Tsymbal, A., Tkachuk, A., Puuronen, S.: Intelligent Medical Diagnostics System Based on Integration of Statistical Methods. In: Informatica Medica Slovenica, Journal of Slovenian Society of Medical Informatics,Vol.3, Ns. 1,2,3 (1996) 109-114
16. Thrun, S.B., Bala, J, Bloedorn, E., et al.: The MONK's Problems – A Performance Comparison of Different Learning Algorithms. Technical Report CS-CMU-91-197, Carnegie Mellon University, Pittsburg, PA (1991)
17. Tsymbal, A., Puuronen, S., Terziyan, V.: Advanced Dynamic Selection of Diagnostic Methods. In: Proceedings 11th IEEE Symp. on Computer-Based Medical Systems CBMS'98, IEEE CS Press, Lubbock, Texas, June (1998) 50-54
18. Wolpert, D.: Stacked Generalization. Neural Networks, Vol. 5 (1992) 241-259

The Notion of 'Classes of a Path' in ER Schemas

Junkang Feng and Malcolm Crowe

Department of Computing and Information Systems, University of Paisley, High Street, Paisley PA1 2BE, United Kingdom
{feng, crow}-ci0@paisley.ac.uk

Abstract. In Entity-Relationship (ER) modeling connection traps are a known problems. But the literature does not seem to have provided an adequate treatment of it. Moreover, it seems to be only a special case of a more fundamental problem of whether a piece of information can be represented by a database that is specified by an ER schema. To develop a systematic treatment for this problem, in this paper we suggest adopting a semiotic approach, which enables the separation of topological connections at the syntactic level and semantic connections, and an examination of the inter-relationships between them. Based on this, we propose and describe the notion of 'classes of a path' in an ER schema, and then indicate its implications to ER modeling.

1 Introduction

Connection traps are a known problem in database design [1], [2], [3], [4], [5], [6], [7], [8]. It would seem that Codd coined the term in the 1970s when he was proposing operations on a relational schema [1], which is then endorsed or followed by Date [2], Cardenas [3] and ter Bekke [4] among others. Then Howe extended this concept to ER modeling in the 1980s [5], which is kept by Howe himself and followed by many well into the 1990s [6], [7], [8]. But we would argue that this problem is not yet thoroughly understood, and thus requires more study. Furthermore, it is only a special case of a more fundamental problem of whether a piece of information can be represented by data in a database.

An ER schema defines the structure of a database. A database is built to store data. From data the user of the database derives information. A common problem in constructing an ER schema, in deriving information by querying a database, and in entering data into a database in order for it to be able to represent certain information is that a database is mistakenly taken to be able to represent or to have represented some information that it is unable to represent. This includes misinterpretation of the meaning of some particular types of connection structures in an ER schema. This is what is meant by the term *connection trap* in the literature. Therefore we suggest that this problem be tackled by looking at the relationship between the information to be represented by a path and the structure of the path.

This paper is organised as follows. In the next section, we will discuss problems with the available treatment of connection traps in the literature and then put forward the concepts of *topological connection* and *semantic connection*. By using these

Eder et al. (Eds.): ADBIS´99, LNCS 1691, pp.218 -231, 1999.
© Springer-Verlag Berlin Heidelberg 1999

concepts, connection traps will be generalised into *connection problems*. In the third section we will describe the notion of 'classes of a path' and define seven different classes that a path possibly has with regard to different sets of semantic connections. In that section, we will also indicate the implications of this notion to ER modeling. Due to the length constraint of the paper, that part has to be brief.

2 Semantic Connections versus Topological Connections

In this section we will show that currently available treatment of connection traps should be extended and generalized. One way of doing so is to adopt a semiotic approach, whereby the notion of *semantic connection* and *topological connection* can be developed.

Codd [1], Cardenas [3] and ter Bekke [4] only deal with connection traps within the context of a relational schema, and Date [2] both a relational and an ER schema. They define them as 'the lack of understanding of a relational composition' [1], 'false inference' [2], and 'represent a ternary relation as two binary relations' [3]. It is regarded by ter Bekke as an intrinsic 'deficiency of the relational theory' [4].

Howe extends this concept to tackle some ER modeling problems and classifies them into 'chasm traps' and 'fan traps' [5]. The cause for a chasm trap is said to be 'partial participation constraint' and that for a fan trap 'a structure of many:1/1:many' [5], [6], [7], [8].

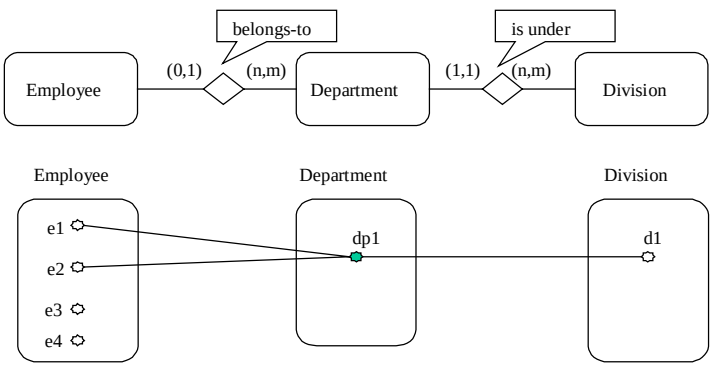


Fig. 1. Partial participation constraints and chasm traps, and false and undesirable transitive connections

But these are not the whole story. We will examine chasm traps in detail to make our points. When a path of length >1 includes one or more *partial participation constraints*, it is possible that at the 'type' level the path suggests the existence of a relationship between entities, but no path actually exists between certain entity instances. When this is not recognised, we fall into a chasm trap. So it could be seen that the cause of a chasm trap were some partial participation constraints, such as the one that entity Employee has in Fig. 1. However, a partial participation does not

necessarily cause a chasm trap. For example, if the path in Fig. 1 were only used to represent the relationship between a division and an employee who belongs to a department and the department is under the division, then the partial participation would be irrelevant. As a result the partial participation of entity Employee would not cause a chasm trap. So whether or not one or more partial participation constraints give rise to a chasm trap depends on the information that the path is used to represent as well as the structure of a path.

This is to say, when considering connection problems, not only should the structure of a path be considered, but also what information the path is used to represent should be taken into account. In fact, falling into a chasm trap is one type of ER modelling problems where a path is interpreted as representing some information that the path is not capable of representing. To this end, it seems helpful to use the terms 'topological connection' and 'semantic connection' to analyse connection problems in an ER schema. This approach is based upon the ideas of semiotics [9] [10], according to which a database can be viewed as a sign system. And the information that is represented by a database is some properties of the signs in the system. Therefore the ER schema for a database can be analysed at two different levels – syntactic and semantic. The former is concerned with the formal structure of an ER schema, and the latter objects and relationships among them that the signs (i.e., data) and structures of the signs signify. Connection problems in data schema design or misinterpretations can be viewed as discrepancies between the two levels.

It can be seen that there are two types of connections between entity instances. The connections that are made possible by the topological structure (i.e., a syntactic level formation of signs) of an ER schema can be termed 'topological connections' regardless of their meaning. The connections in terms of meaning can be called 'semantic connections,' which is the information we want to represent by using some 'topological connections.' 'Semantic connections' are independent of a modelling mechanism such as ER. For example, suppose that 'employee e1 belongs to division d1' is a true proposition, we say that there is a true semantic connection 'belongs to' between two entity instances employee e1 and division d1. If these two entity instances are topologically connected in an ER schema, we say there is a topological connection between them. If the topological connection is identifiable, i.e., the topological connection can be distinguished from the rest of topological connections, and the path in the ER through which the topological connection is made possible can be interpreted as 'belongs to,' then the true semantic connection is said to be represented by the topological connection. To represent a semantic connection by using some topological connection should be a purposeful action – a task in data schema design.

Only when is some semantic connection of an entity instance required to be captured through a path, and yet the semantic connection cannot be represented by any topological connection in the path because the entity instance does not participate in a relationship in the path due to a partial participation constraint, then the partial participation gives rise to a chasm trap.

A chasm trap is a matter of trying to represent (capture) a semantic connection by using a path, but the path is incapable of representing it due to the particular structure of the path. The opposite is that a topological connection through a path is mistakenly taken as representing a true semantic connection. We call such a connection a 'false

and undesirable semantic connection.' A common situation of this kind is what we call 'false and undesirable *transitive* semantic connection.' A transitive connection is a connection that results from two or more connected paths of length 1.

A false and undesirable transitive semantic connection is that a topological path between two instances exists but it is not a true semantic one. We will still use Fig. 1 to illustrate this point. Suppose that the rule 'if an employee belongs to a department, and the department is under a division, then the employee belongs to the division as well' does *not* exist. Then the path should not be interpreted as capturing a 'belongs to' relationship between (even) some employees and some divisions. If we do, we make the mistake of using false and undesirable transitive semantic connections. Note that the structure of the path in Fig. 1 is not of many:1/1:many. This shows that many:1/1:many is not a *necessary* condition[1] for such mistakes. Moreover, this example also shows that fan traps are only a special case of the more general problem of 'false and undesirable transitive semantic connection,' even thought they are probably the most likely ones. We will not describe fan traps in details here as it is not the purpose of this paper. Interested users are referred to [11] and/or [5], the former is more detailed than the latter.

The ideas of connection problems in a path of length >1 can be extended to a path of length 1. In terms of connections between entity instances, the essence of a connection problem is either of the two illustrated in Fig. 2:

- True semantic connections that are taken to have been represented or to be able to be represented in fact are not represented and cannot be represented. The chasm trap falls into this category;
- False and undesirable semantic connections are not recognised. The fan trap is an example of this category.

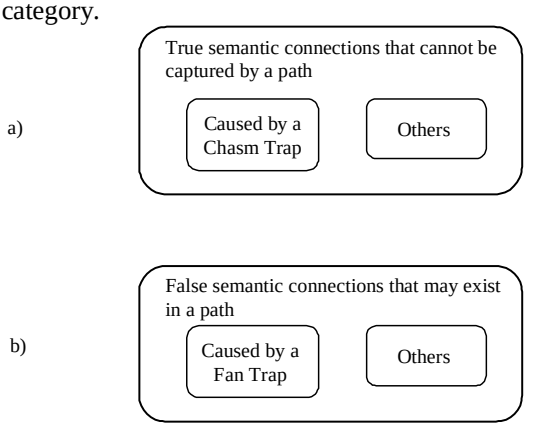


Fig. 2. 'True semantic connections that cannot be captured by a path' and 'false semantic connections that may exist in a path'

[1] The many:1/1:many is not a *sufficient* condition for a false and undesirable transitive semantic connection either. If the 'joint dependency' constraint [2] applies to a path of a many:1/1:many structure or what we call 'total true transitive semantic connection condition' is satisfied [11], the many:1/1:many does not cause a false and undesirable transitive semantic connection including a fan trap.

A path of length 1 can also be misinterpreted in the above two ways. For example, suppose that the 'tutors' relationship in Fig. 4. means 'subject tutoring,' not 'personal tutoring.' If the path were interpreted as 'personal tutoring,' then it would be a misinterpretation as true semantic connections between students and their personal tutors are not and cannot be captured by the path, and the topological connections between students and lecturers captured by the path are undesirable semantic connections with regard to 'personal tutoring.'

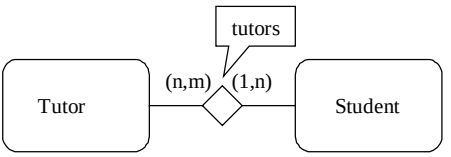


Fig. 3. A path of length 1

Having extended our aforementioned ideas regarding connection problems to cover paths of length 1, we can now classify a path of any length in relation to a given set of true semantic connections (or a given type of true semantic connections) between entities within the path.

3 Classification of a Path in Relation to a Set of True Semantic Connections between Entities within the Path

In relation to a given set of true semantic connections between entities within it, by virtue of set theory and the ideas about connection problems discussed earlier, a path can be classified into one of seven classes. We will describe them one by one. In the discussion, we will use set A to represent a set of topological connections between some entities that are made possible by the structure of a path, and set B to represent a set of true semantic connections between these entities.

Class A (mutual exclusion)

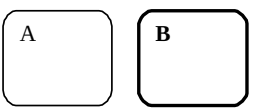


Fig. 4. Class A

$A \cap B = \varnothing$. No member of B is represented by that of A. For example, it seems reasonable to assume that path(supplies, uses) in Fig. 5 captures no information on

‘which supplier is geographically near the headquarters of which projects,’ so in relation to this information, the path is of ‘Class A (mutual exclusion).’

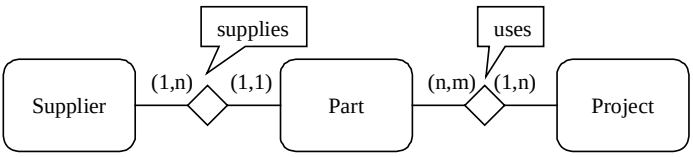


Fig. 5. A path of ‘Class A (mutual exclusion)’ in relation to ‘which supplier is geographically near the headquarters of which projects’

Class B1 (partial and non-differentiable inclusion)

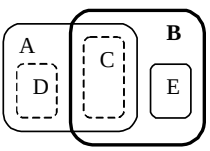


Fig. 6. Class B1

$A \cap B = C \neq \emptyset$, $A \not\subset B$, and $B \not\subset A$. Members of D are not members of B. D are ‘false semantic connections that may exist,’ which may be caused by a fan trap or ‘true but irrelevant semantic connections that can be represented by members of A.’ Members of D and members of C *cannot* be differentiated. E are members of B that cannot be represented by A, which we call ‘true semantic connections that cannot be captured.’ E may be caused by a chasm trap. We will show an example below.

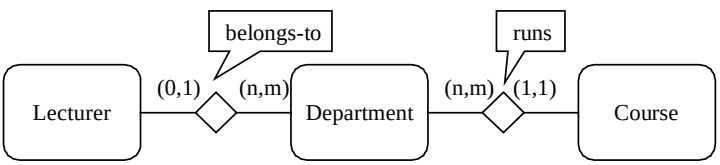


Fig. 7. A path of ‘Class B1 (partial and non-differentiable inclusion)’ in relation to ‘which lecturer is involved in the teaching of which courses’

In relation to ‘which lecturer is involved in the teaching of which courses run by a department,’ path(belongs-to, runs) in Fig. 7 does not capture, say, part-time lecturers’ involvement in the teaching of the courses run by a department, as they do not belong to the department. The path cannot capture ‘which full-time lecturer is

involved in the teaching of which courses run by the department' either, as there is a fan trap in the path. But the path does have some topological connections that represent the correct information, you just cannot tell which ones. So in relation to this information, the path is of 'Class B1 (partial and non-differentiable inclusion).'

Class B2 (partial and differentiable inclusion)

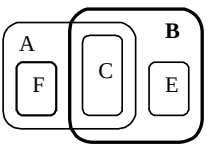


Fig. 8. Class B2

$A \cap B = C \neq \varnothing$, $A \not\subset B$, and $B \not\subset A$. Members of F are not members of B. F are 'irrelevant connections' in A. Members of F and members of C *can* be differentiated. E are members of B that cannot be represented by A, which we call 'true semantic Connections that cannot be captured.' E may be caused by a chasm trap.

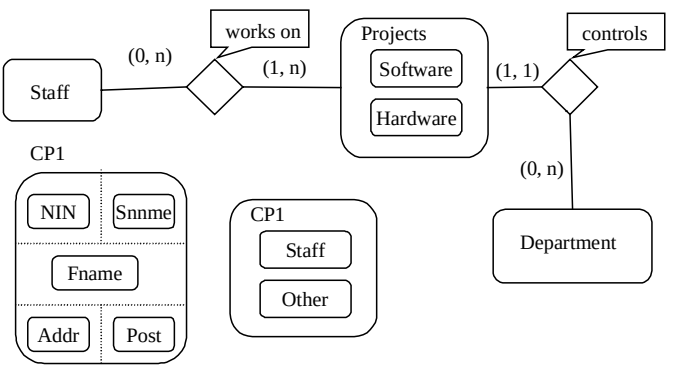


Fig. 9. A path of 'Class B2 (partial and differentiable inclusion)' in relation to 'which engineer works for which department

For example, if there is a rule 'if a member of staff works on a project, and the project is controlled by a department, then the member of staff works for the department,' then path(works on, controls) in Fig. 9 is of Class B2 (partial and differentiable inclusion) with respect to 'which engineer works for which department.' This is because (refer to Fig. 8):

B = 'which engineer works for which department,'
C = 'which engineer who works on a project works for which department,'
E = 'which engineer who does not work on a project works for which department,'

F = 'which member of staff other than engineer who works on a project works for which department.'

B is the union of C and E. A is the union of C and F. C and F can be differentiated by using attribute Post of entity Staff. Note that Fig. 9. is a higraph [12] version of an ER schema. The dotted lines in CP1 indicate that CP1 is the Cartesian product of NIN, Sname, etc., and entity Staff is a subset of CP1. This is a way to capture attributes of an entity in terms of a higraph.

Class C1 (total and non-differentiable inclusion)

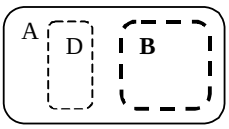


Fig. 10. Class C1

B ⊂ A. Members of D are not members of B. D are 'false semantic connections that may exist,' which may be caused by a fan trap or 'true but irrelevant semantic connections that can be represented by members of A.' Members of D and members of B *cannot* be differentiated. Any member of B has a corresponding member of A. There is no chasm trap. The following is an example.

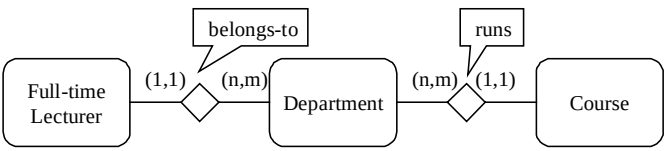


Fig. 11. A path of 'Class C1 (total and non-differentiable inclusion)' in relation to 'which full-time lecturer is involved in the teaching of which courses'

Suppose a full-time lecturer is only involved in the teaching of the courses run by the department he/she belongs to. Then in relation to 'which full-time lecturer is involved in the teaching of which courses run by a department,' path(belongs-to, runs) in Fig. 11. includes all true semantic connections. And yet they are mixed up with invalid undesirable semantic connections due to the fan trap in the path and we are unable to distinguish them. So the path is of 'Class C1 (total and non-differentiable inclusion)' in relation to 'which full-time lecturer is involved in the teaching of which courses run by a department.'

Class C2 (total and differentiable inclusion)

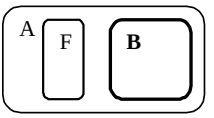


Fig. 12. Class C2

$B \subset A$. Members of F are not members of B. F are 'irrelevant connections' in A.

Members of F and members of B *can* be differentiated. Any member of B can be represented by member(s) of A, i.e., there is no 'true semantic connections that cannot be captured.' There is no chasm trap.

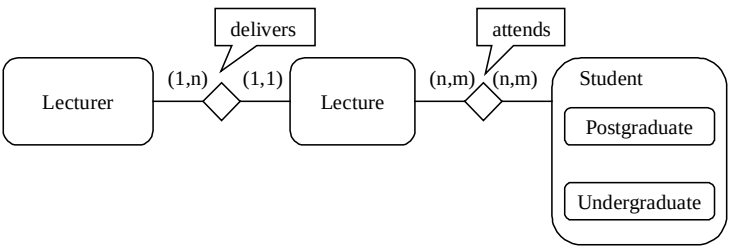


Fig. 13. A path of 'Class C2 (total and differentiable inclusion)' in relation to 'which lecturer delivers which lecture to which postgraduate student'

The topological connections between a lecturer, a lecture and a student that are made available by the path in Fig. 13. are the 'A' in Fig. 12. Notice that the path shows a kind of 'wholeness' and has no fan trap in it. So the 'A' represents 'which lecturer delivers which lecture to which student.' The 'B' is the information that is concerned with, namely 'which lecturer delivers which lecture to which *postgraduate* student.' The 'F' will be ''which lecturer delivers which lecture to which *undergraduate* student.' The union of 'B' and 'F' is the 'A.' And 'B' and 'F' can be differentiated. This enables the path to represent 'B.'

Class D (partial representation)

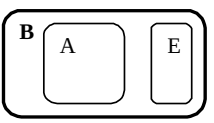


Fig. 14. Class D

A ⊂ B. Any member of A is a member of B, so there is no 'false semantic connections that may exist.' E are members of B that cannot be represented by A, i.e., there are ''true semantic connections that cannot be captured.' E may be caused by a chasm trap.

Suppose that if an employee belongs to a department then the employee always belongs to the division that the department is under. Then all topological connections between employees and divisions through path(belongs-to, belongs-to) in Fig. 15 are true semantic connections. But the path is unable to capture those employees' belonging to divisions who do not belong to a department. So the path is of 'Class D (partial representation)' in relation to 'which employee belongs to which division.'

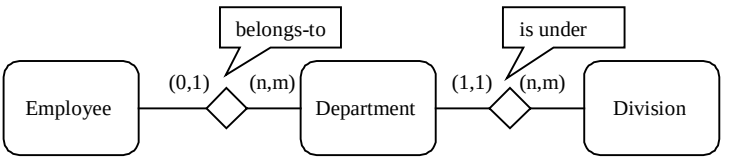


Fig. 15. A path of 'Class D (partial representation)' in relation to 'which employee belongs to which division'

Class E (total representation)

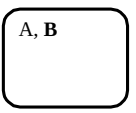


Fig. 16. Class E

A = B. Any member of A represents a member of B, so there are no 'false semantic connections that may exist.' There is no fan trap. Any member of B can be represented by member(s) of A, i.e., there are no 'true semantic connections that cannot be captured.' There is no chasm trap.

In relation to 'which supplier suppliers which parts to which projects,' path(supplies, uses) in Fig. 17. is of 'Class E (total representation),' as all topological connections through the path are about that piece of information, and all that piece of information can be represented by the topological connections made possible by the path.

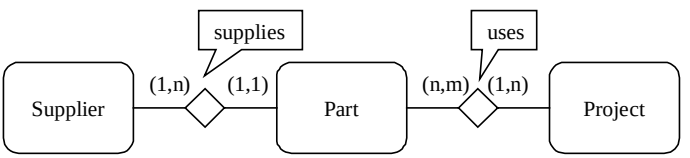


Fig. 17. A path of 'Class E (total representation)' in relation to 'which supplier supplies which parts to which projects'

Note that this is the same path in Fig. 5. where the path is of 'Class A (mutual exclusion),' which is the lowest possible class, in relation to a different set of true semantic connections between the same entities within the path, namely, 'which supplier is geographically near the headquarters of which projects.' This shows the 'relativeness' of the classes of a given path. In relation to different sets of true semantic connections, a given path is normally of different classes.

From the definitions and examples of the classes of paths shown above, we can see that in order to capture a set of true semantic connections between entities, with respect to this set of true semantic connections, one of the following must be found:

- a path of 'Class C2 (total and differentiable inclusion)'
- a path of 'Class E (total representation)'
- a group of paths of 'Class B2 (partial and differentiable inclusion)' or 'Class D (partial representation),' the union of which captures the whole set of true semantic connections.

A deduction can therefore be reached, i.e., if it is of a class other than B2 (partial and differentiable inclusion), C2 (total and differentiable inclusion), D (partial representation) and E (total representation) with respect to a set of true semantic connections, a path is of no use in terms of representing the set of true semantic connections.

The Soundness and Completeness of a Path

To summarize the characteristics of the classes of a path, the concepts of 'soundness' and 'completeness' are useful. We shall define the two concepts as follows:

With respect to a set of true semantic connections between some entities within a path, if all or some distinguishable topological connections that are made possible by the whole path represents at least a proper subset of the set of true semantic connections, then the path is *sound*.

Assume that there are some entities within a path and the entities are mutually exclusive, i.e., the entities are not involved in any structural relationships, namely, a super/subclass relationship. Note that by means of the higraph [12], a structural relationship between entities is represented by blobs (a blob represents a set) and sub-blobs, rather than edges. A blob is a rectangle with round corners in all the figures in this paper. A blob plus its sub-blobs is *not* called a path. Given a set of true semantic connections between these entities, if the path can capture the whole set, then the path is *complete* in relation to this set of true semantic connections. Being 'sound' means being capable of representing at least one true semantic connection, being 'complete' all true semantic connections of a given set. If a path is complete then the path must be also sound. But the reverse is not true, that is, a sound path is not necessarily complete.

The soundness and completeness of a path is related to the question of whether a path is 'sufficient' and 'necessary' for a set of true semantic connections between some entities within the path. We shall use the example in Fig. 18. to illustrate this point.

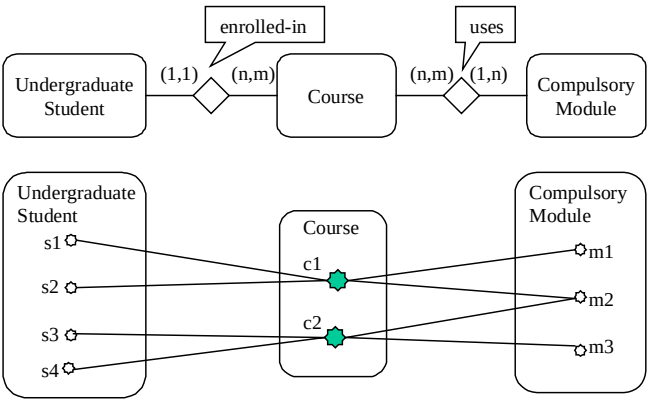


Fig. 18. A path of 'Class E (total representation)' in relation to 'which undergraduate student takes which compulsory modules'

Assume that the following rule holds for the whole ER schema in Fig. 18. :

If an undergraduate student is enrolled in a course then the student will take all compulsory modules of the course. Moreover, if an undergraduate student takes a compulsory module of a course, then there must be a course in which the student is enrolled and the course uses the module. In addition, all undergraduate students are enrolled in courses.

With regard to 'which undergraduate student takes which compulsory modules through being enrolled in which course,' the path in Fig. 18. captures the whole set of true semantic connections, so the path is sound and complete. In other words, if the path is available, then the set is represented. That is, the path is sufficient for representing the set. Thus, a complete (therefore also sound by definition) path is also a sufficient path. And vice versa.

Furthermore, if in an ER schema there is only one sound path with regard to a set of true semantic connections, then in order to represent this set or part of it, the path has to be used; so this path is a necessary path for this set. If a path is sufficient and necessary with regard to a set of semantic connections, then the path and only this path represents the whole set. In other words, the set is represented only once by the ER schema. This is a desirable quality feature of an ER schema.

In Fig. 18, path(enrolled-in, uses) is also necessary with regard to 'which undergraduate student takes which compulsory modules through being enrolled in which course' as there is no any other path at all in the ER schema. It means that in order to represent any semantic connection between some instances of entity Undergraduate student and entity Compulsory module through entity Course, there must be a topological connection of the same instances.

The characteristics of the classes of paths of length > 1 in terms of their soundness and completeness are listed in Table 1.

Table 1. The 'soundness' and 'completeness' of the classes of a path

Class of a path	Soundness	Completeness
Class A (mutual exclusion)	no	no
Class B1 (partial and non-differentiable inclusion)	no	no
Class B2 (partial and differentiable inclusion)	yes	no
Class C1 (total and non-differentiable inclusion)	no	no
Class C2 (total and differentiable inclusion)	yes	yes
Class D (partial representation)	yes	no
Class E (total representation)	yes	yes

Implications for ER Modeling

The notion of 'Classes of a path' can be used to recognize and avoid connection traps. Sometimes it is necessary to restructure an ER schema in order to represent certain information. These two are matters of finding or constructing a sound and/or complete path.

This notion is also powerful and effective for analyzing database transactions at the conceptual level. A database transaction can be seen as either reading a set of semantic connections from the database or changing or setting up (writing) some topological connection(s) in order to represent a set of semantic connections. Reading requires at least a sound path, and writing a complete path.

This notion can and should be extended to cover any components of an ER schema, including specialization, categorization and attributes, whereby to be generalized into the notion of 'Classes of a segment' in an ER schema. This will help the development of a systematic way for examining the information-bearing capacity of a database.

4 Summary

The notion of 'classes of a path' in an ER schema has been proposed in this paper, which can be used to tackle some connection problems in ER modelling. The essence of the problems is whether and how a piece of information is represented by the structure of a database. The approach adopted in this paper is to separate topological connections at the syntactic level and semantic connections, and then examine their inter-relationships.

This paper shows that many types of connection problems in an ER schema can be tackled successfully by using the notion of 'classes of a path.' To examine and avoid connection traps within an ER schema is to make sure that the structure of a database

specified by the ER schema is *sufficient* for representing the information that the database is designed to represent. This is probably the most important objective of database design. This is a matter of the correctness and completeness of an ER schema.

Note that the classes of a particular path is only 'relative' rather than 'absolute,' as with regard to different sets of true semantic connections between objects or different sets of valid objects, a path may be of different classes. This is not a problem, of course, as the usefulness of a path is determined only by the information (in terms of true semantic connections of valid objects) that the path can represent, we should always know what information we want a path to represent. That is, when the class of a path is considered or to be identified, we should already know the context of it.

References

1. Codd, E.F.: A Relational Model of Data for Large Shared Data Banks. Com. of ACM, Vol.13, No.6 (1970), pp.377-387
2. Date, C.J.: Introduction to Database Syatems. 5th edn. Addison Wesley, Reading (1995)
3. Cardenas, A.F.: Data Base Management Systems. 2nd edn. Allyn and Bacon, inc. Boston (1985)
4. ter Bekke, J.H.: Semantic Data Modeling. Prentice Hall. New York (1992)
5. Howe, D.R.: Data analysis for data base design, Edward Arnold. (1983)
6. Howe, D.R.: Data analysis for data base design, 2nd edition, Edward Arnold (1989)
7. Mortimer, A.: Information Structure Design for Databases. Butterworth-Heinemann, Oxford (1993)
8. Conolly, T.M., et al Connolly, T., Begg, C., and Strachan, A': Database Systems - A practical approach to design, implementation and management. 2nd edn. Addison-Wesley Publishing Company, Wokingham, England (1999)
9. Stamper, R.: Organisational semiotics, in Information systems: An emerging discipline? Eds Mingers, J. and Stowell, F. McGraw Hill, London (1997)
10. Andersen, P. B.: A Theory of Computer Semiotics, Cambridge University Press (1997)
11. Feng, J.: An Analysis of Some Connection Problems in an EER Schema by using a Set of 'Info Concepts,' in Computing and Information Systems, M. Crowe ed, Vol 5, No. 2 (1998) pp. 63-72.
12. Harel, D.: On visual formalisms. ACM Communications, Vol 31, No 5 (1988)

Evaluation of Data Modeling

Ronald Maier

University of Georgia, Terry College of Business, Dept. of Management
Athens, GA 30602, USA
rmaier@cba.uga.edu
http://www.cba.uga.edu/~rmaier

Abstract. This paper presents a concept for the evaluation of data modeling which is based on existing theoretical approaches and an empirical study conducted by the author. The main results of this study with respect to evaluation suggest to focus more on organizational issues of modeling, more on process instead of product quality, to consider different application scenarios of data modeling as well as to distinguish the enterprise-wide evaluation of data modeling as an organizational function from the evaluation of single projects using data modeling. The concept consists of recommendations for the evaluation procedure, persons involved, instruments, the design of important organizational dimensions as well as concrete process measures.

1 Introduction and Motivation

During the last 25 years data modeling has become a sound method of developing organizational models both in science and in practice with a strong theoretical basis. The positive effects of its application are hardly questioned in literature. However, it is astonishing that in spite of the abundance of papers on data modeling, up to now hardly any author has focused on quality management of data modeling.

The state-of-the-art concerning the evaluation of data modeling consists of a more or less extensive list of desirable attributes of a data model. It remains uncertain which attributes should be focused in case of conflicts, to which part a single attribute contributes to goals, and in particular how these attributes can be embedded into a comprehensive model for the evaluation of data modeling.

In 1995/96 the author carried out an empirical study on "Benefits and Quality of Data Modeling" (see [9], [11]). The results of this study imply that problems in the quality of data modeling become serious only in more complex environments, especially when employees from different organizational units are involved and various projects using data models have to be integrated.

The goals of this paper are:

- to give a brief survey of approaches for the evaluation of quality in data modeling and
- to present a concept for the evaluation of data modeling based on the results of the 1995/96 study.

Eder et al. (Eds.): ADBIS´99, LNCS 1691, pp.232 -246, 1999.
© Springer-Verlag Berlin Heidelberg 1999

Section 2 motivates a change in the focus of data model quality from product quality to the quality of the process of development and application of data models. Moreover, a classification of enterprise-wide data modeling efforts is proposed based on a matrix model. Section 3 focuses on the evaluation of process quality of an individual project using data modeling. Section 4 summarizes the paper's main findings and presents a brief outlook at the future development.

2 The Relationship between the Quality of a Data Model and Quality of Data Modeling

The term data model denotes either a method for describing data structures (instrument for modeling) or the results of the application of this method (result of modeling) (see [4], 148). In this paper the latter term is used: A data model describes the data items of a certain part of the perceived reality (business domain) relevant for a specific application or a specific user in a structured way. This model includes the relationships between the data. Referring to Juran's general definition of quality ([5], 2.3ff) we consider the quality of data models as the match between the data model and the requirements of its users with regard to the defined application areas of data models. The definition of quality criteria for data models always has to consider the requirements defined by the application context.

The approaches on quality of data models in the literature can be categorized into the following groups: The authors of the first group attempt to develop a comprehensive frame for quality of data modeling (e.g. [8], [6], [15], [13]). The second group develops concrete proposals for the assessment and improvement of individual aspects of data models defining desirable characteristics and criteria for data models (e.g. [1], [2], [3], [12], [18]). The third group does not affect the quality of data models directly. This group either develops approaches or methods to support the process of data modeling the application of which should result in better models.

Up to now the approaches lack integration and agreement among one another as well as with related concepts, such as software engineering or project management. Furthermore these concepts do not take into account the following three points which were found highly important influencing quality of data models in the empirical study conducted by the author:

- different application scenarios of data modeling within organizations;
- organizational design of data modeling and the organizational units involved in data modeling;
- quality of data modeling is not the same as quality of data models.

The author uses these starting points for the definition of his quality concept of data modeling (for a detailed analysis of the related literature and the results of the empirical study see [9], [10]).

The situations which data modeling is used within organizations can hardly be compared in general. The term data modeling has a different meaning in nearly every organization and often the meaning changes even between different units within the same organization. The following ideal application scenarios of data modeling were extracted from comparing different organizations using variables,

which describe the potential and actual use of data modeling for 64 application areas (see [11], 135): data modeling within projects for the development of application systems, support of integration within the IT department, improvement of communication between functional departments and the IT department, data modeling as an organization instrument.

The support of data modeling by methods and instruments is well established and its application within organizations does not cause difficulties. On the other side organizations face serious problems regarding the design of the organizational context for data modeling. But the application of data modeling (and therefore quality) is highly dependent on this context.

Quality of data modeling is not the same as quality of data models. In former approaches to assure and assess quality of data models, the product quality (= quality of data models) was considered almost exclusively. Data modeling as it is used in most organizations is a method, which supports the process of adaptation, standardization and integration in the development of application systems (see, [9]). During this process the individual perceptions and understandings of the members of the organization involved are brought together, discussed, and integrated.

Thus, organizational data modeling is divided into process quality and product quality (quality area). Process quality is the quality of the development and application of data models. Product quality is considered the quality of the results of the modeling process. Moreover we have to distinguish between enterprise-wide assessment of data modeling, and the assessment of an individual project using data modeling (scope). Figure 1 shows the classification of the evaluation into quality area and scope and gives some examples.

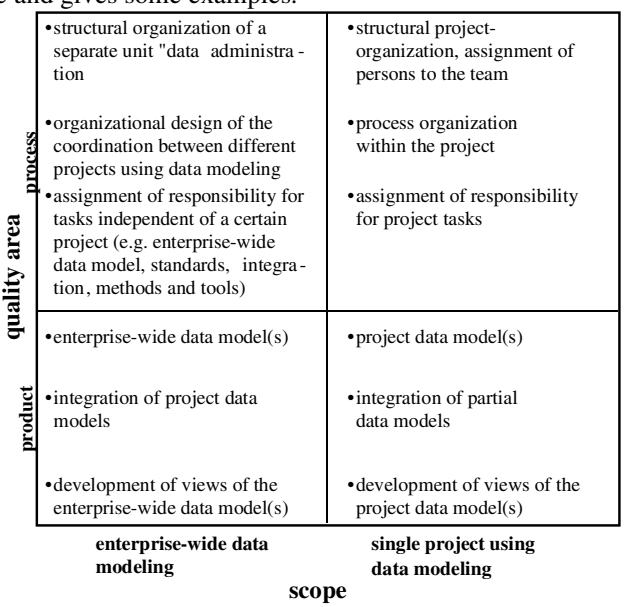


Figure 1: Examples of areas of data modeling to be evaluated according to quality area and scope

Due to the assumed high importance of the processes of developing and applying data models in projects this paper is limited to the evaluation of the process of an individual data modeling project. One might refer to [9] for an assessment of the other three matrix areas.

3 Evaluation of the Process Quality of Data Modeling

According to the results of the author's empirical study, seven organizational dimensions are key factors influencing the quality of data modeling (see figure 2). The "dimensions" represent perspectives on the organizational context of data modeling.

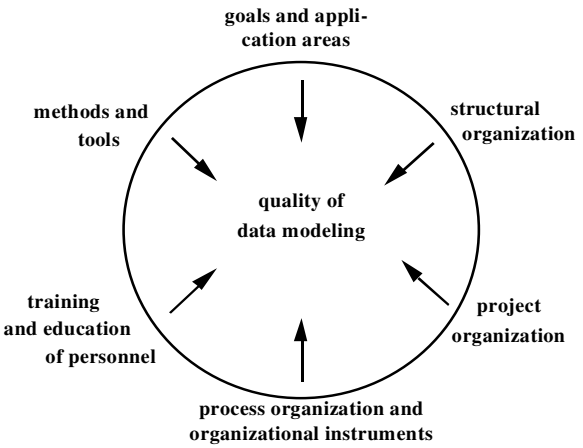


Figure 2: Dimensions of the organizational design of the data modeling environment and their effects on the quality of data modeling

There are interdependencies between the perspectives, which have to be considered. In section 3.1, the approach for the evaluation of the processes and the persons involved are discussed. Sections 3.2-3.7 discuss the evaluation of the process focussing on the dimensions of quality management of data modeling mentioned above. Finally some selected measures are presented (section 3.8).

3.1 Approach and Persons to Be Involved

The following groups of employees can be involved in the evaluation of data modeling projects:

- Members of the project team: e.g. application developers, employees of the functional departments involved, data administrators or experienced data modelers involved in the project e.g. as "consultants," data base developers;
- Members of other project teams who also perform tasks of data modeling;
- Data administrators and experienced data modelers who are not involved in the individual project but who are responsible for assuring quality of data modeling;
- Employees involved in quality control;
- Externals.

The assessment by each of these groups has advantages and disadvantages. The advantage of the assessment by *project members* is that they have the best knowledge of the area to be modeled as well as of the process of modeling. A disadvantage is the missing objectivity concerning the project to be assessed.

In the second case employees of different projects assess each other's data models (a kind of "peer review"). This results in potential advantages concerning the integration of data modeling projects. The data modelers become attentive to the problems of other teams. If the models that have to be assessed are selected appropriately the data modelers gain knowledge about models that are related to their own models. Also they are best suited to evaluate the interfaces and overlaps between their own models and the models that have to be assessed. On the other hand, conflicts between the teams may arise that may lead to inefficiencies.

Data administrators and experienced data modelers can assess the integration and consistency with already existing models very well. This is due to their independent, organization-wide perspective. Moreover, data administrators usually have good knowledge about the rules, regulations and guidelines, the methods and the standards to be considered. A comparably restricted knowledge about the details of the business domain to be modeled can take effect disadvantageously. On the one hand this causes the problem that data administrators do not make any or make only a few comments about the content of the data models. On the other hand the data modelers in the project team may reject their comments and suggestions pointing to the lack of domain knowledge of the data administrators.

Employees involved in quality control know best about the implementation into an integrated organizational or IT-wide quality control system for application development. A disadvantage could be that employees of quality control do not have domain knowledge and are usually not familiar with data modeling.

The assessment by externals bears the advantage of their assumed total independence. In addition there is low risk that the assessment causes conflicts with the assessed employees or groups of employees. On the other hand, it can be a disadvantage that the know-how won in the assessment leaves the organization and could be made available to competitors.

Table 1 shows recommended areas of evaluation for each of these groups. It is recommended to conduct the evaluation not only by one group but to include several groups in the evaluation procedure. The evaluations done by each of these groups are to be compiled and should be explained to the modelers in a meeting. Modelers and evaluators can decide on measures for improvement together.

Evaluating group	area to be evaluated
Members of the project	Representation of the business domain correct and complete, project goals reachable and sensible; project organization useful, co-operation with non-project members involved
Members of other projects	interfaces (e.g. consistency, understandability); overlappings with other project data models correct and useful; co-ordination and co-operation with other teams
Data administrators	re-use and re-usability; ease of integration and consistency e.g. with enterprise-wide or departmental data models; standards, rules and regulations, descriptions complete and understandable
(software) quality assurance	integration of data modeling in the context of application development; comparing the quality of the resulting application system(s) to the quality of the data model(s)
Externals (e.g. consultants)	application of methods and tools for data modeling; structural and process organization

Table 1: Examples for evaluating groups with recommended areas to be evaluated

There is hardly any literature to be found about how to proceed in the assessment. Basically we have to distinguish between tool-based (automated) quality assessment, reviews, and subjective assessment by employees involved in data modeling or by non-members of the project team (e.g. experienced data modelers). In general we will find a combination of all three types. As the availability of tool-based assessment increases the manual inspection concentrates more and more on the content-related criteria and the processes of development and use of data models.

When reviews are used the following guidelines regarding the schedule are suggested (see [15], 28f): At least two reviews should be done independent of the size of the modeling effort. After the final review is done the team should have sufficient time to consider the suggestions for improvement. In more complex modeling projects there should be one review per month. Additionally we suggest to support an analysis of the grade to which the goals are reached. Table 2 shows an estimation of the expected duration of the first review depending on the size of the data model to be assessed.

Size of the model	Estimated duration of the review
50 entity types	≥ 1 day
100 entity types	2-3 days
200 entity types	5-7 days
300 entity types	9-12 days

Table 2: Estimated duration of the first review dependent on the size of the model (source: [15], 27)

The duration of the review is influenced by several factors like goals and application areas of modeling, number and heterogeneity of the persons involved, importance of the model, size of the model, degree of completion, complexity of the business domain, type of the model (strategic or operational), number and experience of the reviewer, and the tools used for the review (see [9], 308f).

3.2 Goals and Application Areas of Data Modeling

In the organization, data modeling is used in several application areas. The organizational setup has to be chosen depending on the goals to be reached in a certain project using data modeling. If the goals and the application areas of data modeling are clearly defined, we expect an improvement of the motivation and the acceptance of all employees involved in the data modeling project, a precise and unambiguous design of the surrounding organizational dimensions as well as the avoidance of unproductive conflicts between the employees involved.

Goals, importance, and application areas of data modeling can be stated in data modeling guidelines. Data modeling has to be directly related to modeling other views of the organization (e.g. functional or process modeling). The guidelines could provide information about these areas:

- Goals of data modeling: IT related goals (e.g. integration of application systems, support in the selection of standard software), goals related to functional departments (e.g. support of the understanding of data structures, extension of the data volume that can be requested autonomously by functional departments), goals of the top management (e.g. improvement of the decision support by management information systems and online analytical processing);
- Persons and person groups who are to be involved in data modeling, clear definition, restriction and coordination of roles of different groups involved (e.g. functional departments, application development, data administration) both during the development and use of data models;
- Interdependence to IT strategy or to organizational strategy: Significance of the data in the organization and the contribution to the achievement of IT-related goals expected from data modeling (e.g. improvement of the time needed to response to requests of business partners by integration of application systems which is supported by data modeling);
- Related areas: to the modeling of other views, to integration of data modeling into a "data life-cycle", and to application system development;
- Assurance of the support of the project by top management, by defining the "data modeling concept" obligatorily.

It is important that all participants and the representatives of all involved groups know about the guidelines. The ideal application scenarios identified in the empirical study can be used during the definition of the application areas.

3.3 Structural Organization

First we need to determine areas of responsibility regarding data modeling and then we need to assign these responsibilities to the employees and groups involved in data modeling. Responsibility for data modeling can be assigned to functional de-

partments, application areas, projects to develop application systems, or data modeling tasks. Most times a combination is reasonable.

Problems with the assignment of responsibility can result especially if employees from different organizational departments co-develop a data model or data models must be integrated. Table 3 provides a proposal for a possible assignment of responsibility to selected groups of employees. An evaluation can be based on this proposal.

A decision about the establishment of a separate organizational unit for data related tasks (e.g. data management) depends on the role of data within the organization. Basically we can say: The more varying and the more extensive the application areas of data modeling are,

- the more data modeling is used for integration and standardization (not only within the IT area),
- the more employees are involved in data modeling,
- the more employees with different professional experience are involved in data modeling,
- and the more data models are used,

the more reasonable it is to create a separate organizational unit for data related tasks.

area of responsibility	**functional dept.**	**Application developm.**	**data management**	**methods tools**
selection of methods/tools	C	C	C	D/Ca
support for problems with methods/ tools			Ca	Ca
observation of organizational norms		Ca, E	D, E	
observation of external norms		Ca, E	D, E	
(functional) content	D/Ca, E	C	C, E	
(structural) content		D/Ca, E	C, E	
resolving conflicts according to contents			D/Ca	
integration of partial data models with regard to contents	C	C	D/Ca	
integration with other parts of the systems design		D/Ca	C	
training and education			D/Ca	
use of data models	Ca	Ca	Ca, E	

Table 3: Recommendation for assigning responsibility for data modeling to groups of employees (legend: C=consulting, Ca=carrying out, D=decision, E=examination)

3.4 Project Organization

It is not possible to discuss general findings about project organization within this paper (see [7], 463ff). Here are some characteristics of the organization and suggestions particularly for the management of data modeling projects:

Team Formation and Team Composition

As shown earlier, there should be project-independent rules or guidelines available regarding how to form a data modeling team. The team members are to be recruited and their roles in the team are to be determined. The need for training has to be analyzed. The qualifications of potential team members can be compared to a standardized profile for data modelers. It seems to be important to determine all employees involved in data modeling who are not members of the data modeling team ("stakeholders"). This could be the heads of the departments from which employees are sent into the project team or specialists who will work with an application system to be implemented. The project related goals and the relation to their work are to be pointed out to them in order to achieve acceptance and support for the project. Furthermore, regular information can keep the stakeholders up to date about project status and allow for feedback.

Integration of the Functional Departments

There are several possibilities to involve functional departments in the development of data models. Up to now we discussed the role of certain employees from functional departments as regular members of the project team. Further possibilities are (in the order of the frequency they appear in the organizations I surveyed): meetings, interviews, feedback sessions, document analyses, workshops, prototyping and questionnaires.

A general statement about which form of involvement of the functional departments is the best seems impossible. In the case of projects to develop application systems in limited areas with unambiguous requirements, and after consultation of the functional departments, application developers can do data modeling without involvement of other members of the organization. This does not necessarily cause quality problems and it results in lower costs.

In those cases in which data modeling focuses on standardization across departmental or even organizational borders, harmonization of interfaces, and is used to reach agreements between the involved project members a combination of workshops with altering members, feedback sessions, and – if application systems are to be developed – prototyping is considered the best solution.

3.5 Process Organization and Organizational Tools

The design of processes should be based on the methods used for data modeling. In some methods, a phase oriented procedure is proposed (see e.g. [14], 34, [17], 164). However, most "methods" for data modeling are limited to structuring data and the graphic representation of data and data requirements (a notation). The description of

an ideal process, which a project using data modeling should follow, is necessary in order to coordinate the work of employees involved in projects.

In order to guarantee the subsequent use of data models during implementation and in order to ensure the actuality of data models in a changing organizational environment the adjustment of parts of the data models (e.g. individual data elements) might be necessary. A central data dictionary ensures the actuality of data definitions. Corrections in the central data dictionary should be made only by certain persons at specified times (unfreezing-freezing). The test of modifications should guarantee that no unwanted side effects occur.

Organizational instruments are used to state and document the decisions taken concerning organizational design of data modeling. The decisions are institutionalized in method manuals, tool manuals, guidelines for enterprise-wide standardization of data elements, organization specific rules for data modeling (e.g. for the definition of names, for the use of elements of data modeling that are not provided in the methods used, for the integration and merging with other partial models, and for the definition of the process), guidelines concerning the documentation of the history and the implementation of elements of data models, and guidelines for quality control.

In organizations in which several data modeling projects with different goals occur at the same time a multilevel design of the organizational instruments e.g. in organization-wide guidelines, guidelines for specific application areas, and guidelines specific for user groups are recommended. In addition to rules, guidelines and manuals, a part of coordination of data modeling can be covered by consistent application of the functions provided by data modeling tools.

3.6 Personnel Training and Education

There is limited information about the necessary education and professional experience of data modelers to be found in literature. The description of functions developed by the Swiss Union for Data Processing can be used as a source for the generation of job profiles (see [16], in particular the job description of data adminstrator, data base expert I-III as well as data architect I-II).

After an introduction to the goals and application of data modeling has been taught during the initial education a sense for data modeling should be communicated by means of easy examples which show that different data models (e.g. concerning the name selection, the structuring, and the necessary degree of detail) can represent an "optimal solution" depending on the context of modeling.

After that negtiation and agreement between several groups of employees should be simulated in case studies or role-plays. This demonstrates the most important challenges of data modeling and which approaches can be used to solve related problems.

Training and education should be designed with respect to the different groups of employees. Table 4 shows the knowledge assumed to be necessary for data modelers according to the employees' position in the organization. It has to be considered that different kinds of knowledge are important depending on the application situation.

The responsibilities could be distributed as shown in Table 3. Data management plays the role of a coordinator in the development of data models.

content of education and training	functional dept.	application developm.	data mgmt.
dm-methods	4	5	5
dm-tools	3	5	5
db-systems	1	5	3
db-theory	1	5	3
communication	4	4	5
systems analysis & development	1	5	3
project management	1	3	5
business domain	5	3	3
quality management	5	5	5
didactics	1	1	5
organizational design	3	3	5
security	5	5	5

Table 4: Content of education training and their priority according to groups of employees (5 = highest, 1 = lowest priority)

3.7 Methods and Tools

Methods and tools support the tasks of data modeling. They define a framework for the process and for the participation of employees and they force them to stick to the rules and guidelines. Thus, they standardize the development of data models. Methods and tools have a significant impact on process and product quality of data modeling (see [17], 163).

Information about requirements for methods and instruments of data modeling can be found in [9]. The selection of methods and tools should not be done before the application context has been determined and the organizational framework of data modeling has been designed.

Groupware tools and information sharing and communication applications based on Internet technologies (Intranet) can be used to support communication, coordination, and cooperation during the process of data modeling. Teamwork in data modeling projects is characterized by workshops or sessions, which can be supported by these technologies. Furthermore, it is conceivable to have virtual teams working synchronously and asynchronously on problem solutions in different locations.

The use of reference data models should have positive effects on the costs for development and maintenance of organization-specific data models. We expect

potential benefits for training and education, development and maintenance of data models from the application of reference data models (see [2], 32ff).

When deciding about the use of a reference data model, the costs of the model have to be compared to the benefits. Due to strong regulations by law or by other requirements many organizations use similar data structures (e.g. financial accounting, wage processing). In these organizations the use of reference data models can have great advantages. In areas in which organizations try to differentiate from their rivals the application of reference data models is not recommended. In particular it has to be considered that not only the resulting data models, but in particular the process of development and use of the data models generates benefits. This can be supported by the use of reference data models but not replaced.

3.8 Measures of Process Quality

The assessment of development and application of data models up to now is not supported by measures. It is important, however, to gain experiences about the effect of different configurations of the organizational dimensions in order to be able to estimate the quality of data modeling in a better way.

Expenses/Productivity

To give an estimation of productivity e.g. the expenses of the development, the number of developers and the number of persons involved in a data modeling project have to be recorded. This has to be done separately for the different groups of employees e.g. functional departments, application development, data administration and the group responsible for methods and tools. The costs of data modeling are to be assessed according to the following areas:

- Expenses for employees who are responsible exclusively for data modeling (e.g. data administrators);
- Expenses for team members, separate for the groups of employees (e.g. functional departments, application development);
- Expenses for external experts (e.g. experts who support the introduction of data modeling in the organization);
- Expenses for hardware and software: e.g. data modeling tools, workstations, communication system, printers);
- Current expenses for the operation of the data modeling environment: e.g. repair costs, costs for support by developers of the tools, paper, transfer costs, costs for the use of computers (mainframes);
- Expenses for initial education and on-going training.

The costs determined can be compared with the expenses for the overall software or system design:

The number of elements developed per unit of time can measure the productivity of data modeling. These elements can be entity types, relationship types or attributes depending on the type of data model (e.g. architecture or detailed model). We have to differentiate between productivity of the development of data models (PNEW) and the productivity of modification and extension of data models (PCHANGE).

Additionally the number of conflicts about the meaning of terms solved and the number of standardized terms per unit of time can be used as dimensions of productivity.

$$PNEW(m) = \frac{Number\ of\ newly\ developed\ entity\ types\ (or\ relationship\ types,\ attributes)\ (m)}{Number\ of\ person\ hours\ (m)}$$

$$PCHANGE(m) = \frac{Number\ of\ changed\ entity\ types\ (or\ relationship\ types,\ attributes)\ (m)}{Number\ of\ person\ hours\ (m)}$$

Variability

$$Variability(m) = \frac{Number\ of\ changed\ elements}{Number\ of\ elements\ *\ Time}$$

Modeling environment

This examines the relationship between the data model and an implementation into an application system or a data base system.

$$Implementation1(m) = \frac{Number\ of\ entity\ types(m) + Number\ of\ relationship\ types(m)}{Number\ of\ data\ base\ tables(db[m])}$$

$$Implementation2(m) = \frac{Number\ of\ attributes\ (m)}{Number\ of\ data\ base\ attributes(db[m])}$$

Complexity of the Application Situation

The complexity is affected by the number, the variation and the hierarchical classification of the involved groups of employees, the type, the number and the variation of the application areas, and the type of model: strategic versus functional data model and the estimated grade of newness of the project (ratio of parts from other data models reused to newly modeled parts). The complexity of the modeling project can be related to the expenses.

4 Summary and Outlook

What is quality of data modeling? How can quality be assessed and assured? These questions cannot be answered easily. Due to the varying nature of data modeling no general concept for the evaluation of data models can be given. This paper presents recommendations for the evaluation of important organizational dimensions, which influence the quality of data modeling. These recommendations have to be related to certain application situations and to the goals of data modeling. Ideal application scenarios were used to show the varying nature of the application of data modeling.

Moreover, it is not the quality of data models but the quality of data modeling that has to be focused. The paper suggests a categorization into process and product

quality for the assessment of data modeling as well as into single data modeling projects and enterprise-wide data modeling. A part of the expenses of data modeling is not related to projects to develop application systems but is a part of the enterprise-wide data management and its long-term goals. Thus, it is part of the infrastructure and has to be separated from other expenses of data modeling.

Put in a nutshell, the evaluation of data modeling has to concentrate on process oriented quality criteria and it has to take into account an extension of the focus to be evaluated and varying application scenarios.

5 References

1. Batini, C., Ceri, S., Navathe, S. B.: Conceptual Database Design: An Entity-Relationship Approach. Redwood City et al. 1992.
2. Hars, A.: Referenzdatenmodelle. Grundlagen effizienter Datenmodellierung. Wiesbaden 1994.
3. Heilandt, T., Kruck, P.: Ein algorithmisches Verfahren zur Bewertung und Verdichtung von Entity-Relationship-Modellen. In: Informatik-Forschung und Entwicklung 8 (1993), S. 197-206.
4. Heinrich, L.J., Roithmayr F.: Wirtschaftsinformatik-Lexikon. 4th ed., Munich, Vienna 1992.
5. Juran, J. M., Gryna, F. M. (ed.): Juran´s Quality Control Handbook. 4th ed., New York et al. 1988.
6. Krogstie, J., Lindland, O.I., Sindre, G.: Defining Quality Aspects for Conceptual Models. In: ISCO 3 - Proceedings of the International Conference on Information System Concepts - Towards a Consolidation of Views, Marburg 1995.
7. Lehner, F., Auer-Rizzi, W., Bauer, R., Breit, K., Lehner, J., Reber, G.: Organisationslehre für Wirtschaftsinformatiker. Munich, Vienna 1991.
8. Lindland, O.I., Sindre, G., Sølvberg, A.: Understanding Quality in Conceptual Modeling. In: IEEE Software 11 (1994) 2, 42-49.
9. Maier, R.: Qualität von Datenmodellen. Wiesbaden 1996.
10. Maier, R.: Benefits and Quality of Data Modelling - Results of an empirical analysis and definition of a framework for quality management, in: Thalheim, B. (ed.): Conceptual Modeling - ER'96 - Proceedings der 15th International Conference on Conceptual Modelling ER '96 in Cottbus (Germany), 1996, 245-260.
11. Maier, R.: Nutzen und Qualität der Datenmodellierung - Ergebnisse einer empirischen Studie. In: WIRTSCHAFTSINFORMATIK, Vol. 40, Nr. 2, 1998, 130-140.
12. Marche, S., Measuring the Stability of Data Models. In: European Journal of Information Systems, Vol. 2, No. 1, 1993, 37-47.

13. Moody, D.L., Shanks, G. G.: What Makes a Good Data Model? Evaluating the Quality of Entity Relationship Models. In: Loucopoulos, P. (ed.): ER '94 - Business Modeling and Re-Engineering, Proceedings of the 13th International Conference on the Entity-Relationship Approach, Manchester, GB, Berlin et al. 1994, 94-111.
14. Ortner, E., Söllner, B.: Semantische Datenmodellierung nach der Objekttypenmethode. In: Informatik-Spektrum 12 (1989) 1, 31-42.
15. Reingruber, M. C., Gregory, W. W.: The Data Modeling Handbook. A Best-Practice Approach to Building Quality Data Models. New York 1994.
16. SVD - Schweizerische Vereinigung für Datenverarbeitung, VDF - Verband der Wirtschaftsinformatik-Fachleute (ed.): Berufe der Wirtschaftsinformatik in der Schweiz. Zürich 1993.
17. Szidzek, A.: Datenmodellierung - Vorgehensmodell zur Konstruktion und Einführung einer unternehmensweiten, konzeptionellen Datenstruktur. Würzburg 1993.
18. Zamperoni, A., Löhr-Richter, P.: Enhancing the Quality of Conceptual Database Specifications through Validation. In: Elmasri, R., Kouramajian, V., Thalheim, B. (ed.): Entity-Relationship Approach - ER '93, Proceedings of the 12th International Conference on the Entity-Relationship Approach in Dallas-Arlington (USA), Berlin et al. 1994, 96-111.

Organizational Modeling for Efficient Specification of Information Security Requirements

Jussipekka Leiwo *, Chandana Gamage, and Yuliang Zheng

Monash University, PSCIT
McMahons Road, Frankston, Vic 3199, AUSTRALIA
Phone +61-(0)3-9904 4287, Fax +61-(0)3-9904 4124
{skylark,chandag,yuliang}@pscit.monash.edu.au

Abstract. Functional security requirements of information systems can roughly be classified into two: computer security requirements and communications security requirements. Challenges for developing notations for expressing these requirements are numerous, most importantly the difficulty of dealing with layers of abstraction, flexibility to adapt into many types of requirements, groupings of requirements, and requirement dependencies. Many frameworks for dealing with information security highlight the importance of a properly defined organization of security but fail to establish models to support the specification. This paper establishes one such model and demonstrates how the above difficulties can be overcome through extensive application of organizational modeling of information security.

1 Introduction

1.1 Background and Motivation

Typical information security requirements in a single level, general purpose, stand-alone computer system are expressed in terms of access control. Assuming a user is properly authenticated, which access does a user or programs executing on behalf of a user have on different resources. The access matrix by Lampson [15] provides a framework for specifying and enforcing access control requirements in such environments. Most commercial operating systems implement a variant of this. Security requirements in multilevel secure (MLS) environments are more complicated. Access control has to be expressed as mandatory (MAC) and discretionary (need to know) access control(DAC) requirements. Bell and LaPadula [6] established a theory for determining access to protect confidentiality in MLS systems based on the military hierarchy of information classification. Other security requirements in MLS systems can be expressed in terms of non-interference [12], non-deducibility [26], and hook-up property [20].

* From Sept. 1, 1999, author has been with Vrije University, Division of Mathematics and Computer Science, Faculty of Sciences, De Boelelaan 1081a, 1081 HV Amsterdam, The Netherlands, leiwo@cs.vu.nl

Eder et al. (Eds.): ADBIS'99, LNCS 1691, pp. 247–260, 1999.
© Springer-Verlag Berlin Heidelberg 1999

In commercial systems, protection of data when communicated over insecure networks is more relevant. Access controls are typically expressed as work roles, and individuals are then assignment to different roles depending on the specific task they use the system for [23]. Other security requirements in distributed systems govern confidentiality, integrity, authenticity, authorization and non-repudiation [10], typically achieved through applications of cryptography. Confidentiality of communication is the basic application of encryption of messages. Integrity and authenticity are achieved by digital signatures, message authentication codes and other cryptographic integrity checks. Access control is usually specified at system access level by, for example, firewalls and screening routers, and standard operating system authorization mechanisms at the object level. Evidence for resolving disputes in case of non-repudiation is gathered and protected by cryptographic measures.

These protection requirements are strongly related as illustrated in Fig. 1. Together they are set to achieve the common security objectives, protection of confidentiality, integrity and availability of data. Most cryptographic applications require a number of cryptographic keys and the security is solely dependent on the key. This is assuming that the algorithms and protocols are adequately specified and implemented. Therefore, special care must be exercised when dealing with cryptographic keys. Key generation, key exchange and key storage are common indirect requirements for any cryptosystem.

In addition to the hierarchy of security objectives, security services and security mechanisms, a number of additional groupings of security requirements have emerged. The most important one is the one of the Common Criteria for Information Security Evaluation [8]. Common Criteria (CC) establishes a process for evaluating the security of information systems. Security requirements, called security function specifications, of systems are derived from abstract security objectives. These requirements are grouped into security classes. Each class then consists of a number security families. Security families consist of a number of components, that are finally the actual specifications of security functions. Therefore, any mechanism to deal with information security requirements must support grouping and dependencies of requirements on such a level of abstraction that provides independency of the underlying grouping scheme.

The derivation of functional security requirements from abstract security objectives is a crucial task in information security [25]. Adequately defined information security organization representing different views towards information security is a fundamental success factor in design, implementation and operation of information security [3]. As LaPadula [16] states, this is one of the areas of information security where extensive research is required. Many models for information security focus on the process of dealing with information security requirements, not on the organizational dimension.

The model of Anderson, Longley and Kwok [1] focuses on the identification and evaluation of various threats originating from operational environment and systems that the assets under protection must face. Organization in which this work is done is not considered. Most other models for information security de-

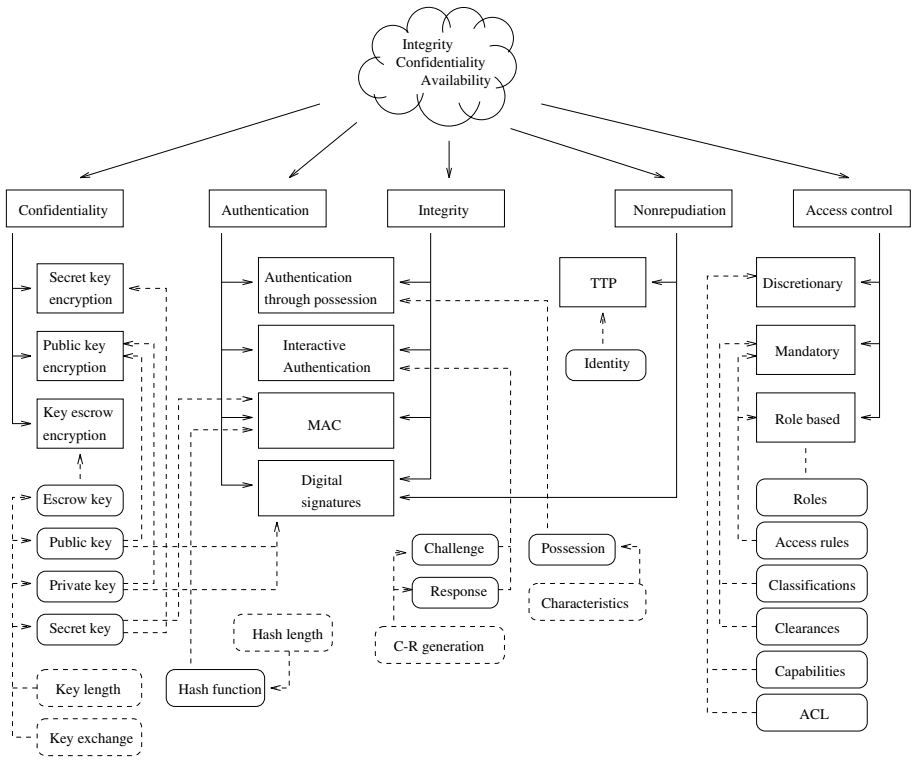


Fig. 1. Dependencies of information security requirements

sign also focus on identification and evaluation of vulnerabilities of systems and specification of countermeasures to these vulnerabilities [24,28,22]. The significant implication of these methodologies is acknowledgment of risk analysis as the foundation of information systems security. This has been a common assumption since early works by Parker [21] and Fisher [9] but has recently attracted considerable amounts of criticism. Because of the limitations of theory of probability in capturing the complexity of information risks [2], the cost of risk analysis and high number of subjective modeling decisions required [27], and small scientific value of risk analysis [4], proposals have been made for alternatives for risk analysis.

Baseline protection manuals and extensive check lists, such as [7] and [14], have common problems with risk analysis. Baskerville [5] has identified risk analysis and checklists belonging to most primitive category of tools for designing security measures for information systems. Therefore, applications of these mechanisms are unlikely to significantly advance scientific knowledge in the design of information security. Rather, models should be develop to logically model information security in organizations. Backhouse and Dhillon [2] have made an attempt to model information security as a structure of responsibility and duty

but their model anyhow, remains on a considerably high level of abstraction and does not provide pragmatic methods for dealing with information security in organizations.

1.2 Contribution of the Paper

The major contribution of this paper is specification of a model that focuses on the modeling of the organization in which information security is developed. This enables simple notations for expressing information security requirements since hierarchies and dependencies of requirements are handled through organizational modeling, not through requirement syntax. The proposed mechanism also aids in the advancement of scientific knowledge on the management of information security by enabling formal modeling of information security in organizations. This is especially important since latest trends in the management of information security suggest returning from risk analysis to extensive check lists.

This paper extends the theoretical foundations established in [18]. Specification of a simple notation for expressing information security requirements in [19] transfers the focus of information security design to the adequate modeling of organization instead of actual information security requirements. The organization is a hierarchy to clearly illustrate different levels of abstraction required in comprehensive information security, and is therefore on align with many generic principles for the organization of information security. In addition to the specification of a mechanism for establishing an information security development organization and a syntax for expressing information security requirements, the framework under discussion consists of harmonization functions and merging of requirements. Harmonization functions are a construct to express organizational knowledge of desired state of security and to formally enforce this knowledge in requirements. Merging is a function to combine two requirement bases, i.e. collections of requirements assigned to a specific component in the information security development organization, in a manner that identifies and solves conflicts there may be between the two requirement bases. Harmonization functions and merging of requirement bases are of little value for understanding the role of organizational modeling in security development. Therefore, and due to the lack of space, their existence is assumed and only a brief summary is provided. Consultation of [17] is recommended for full details.

A software tool has been developed to aid in the applications of the theory discussed in this paper, and is publicly available[1] for evaluation. The software has been applied in the case study in medical informatics application domain, and some of the findings shall be documented in this paper.

1.3 Note on Terminology

The theory being discussed is called harmonization of information security requirements. This is because an information security development organization

[1] http://www.pscit.monash.edu.au/links/harm/

is divided into layers, each representing different level of abstraction of information security requirements. Each layer consists of a number of units that are the actual components representing responsibility of information security in an organization. These units then process local security requirements and provide a harmonized set of information security requirements for the units at lower levels of abstraction to further modify and add details into requirement primitives. The hierarchy is characterized by *Child* and *Parent* relations describing the way information security requirement primitives are passed between layers of abstraction. Therefore, the structure of information security development organization may be very different from the way business is organized. Layers and units in an organization represent responsibilities regarding information security, not general business responsibilities in an organization.

Information security requirement is a highly abstract concept used to represent specifications of security measures at all the levels of abstraction. By the harmonization of information security requirements, the abstraction is reduced step by step to derive concrete security specifications from abstract information security objectives. Information security objective is an informal expression of intent to establish security. Information security requirement at highest level of abstraction is a formally expressed full or partial information security objective. The initial task of security design is to develop and organizational information security objective and express that objective in a formal manner. This expression is the initial set of information security requirements. It is assumed that objective is correctly expressed. The mechanisms demonstrated in this paper is only capable of resulting in adequate security if initial requirements and additional organizational knowledge are correctly expressed. The strength of the proposed approach is the simplicity of both tasks, believed to reduce the likelihood of errors, but direct assurance from correctness of expressions can not be provided.

1.4 Structure of the Paper

Section 2 first establishes the approach towards modeling of an information security development organization. Section 3 then introduces the notation for expressing information security requirements. Grouping of requirements and requirement dependencies are studied in Sect. 4 and 5. Section 6 summarizes some findings of a case study. Conclusions are drawn and directions highlighted for future work in Sect. 7.

2 Modeling of the Information Security Development Organization

Information security development organization can be modeled as a hierarchy [18]. The organization consists of a number of layers and each layer consists of a number of units. Layers represent levels of abstraction towards information security, and unit represents a single point of responsibility regarding information security. At high levels of abstraction, a unit can represent responsibility of

information security in an entire organization and at lower levels of abstraction responsibility of specific subcomponent of a security enforcement in a computer system. Theoretically, units can represent also international and national laws, agreements, treaties and other documents coordinating information security in organizations, but such levels of abstraction are more a concern of the specification of organizational security policy objectives, and therefore outside of the scope of practical applications of the harmonization of information security.

Each unit and layer consists of a separate requirement base. Requirement base is a collection of requirements assigned to that layer or unit. Final requirement base of a unit is composed by combining upper layer requirement primitives and layer specific requirements with original requirement base of a unit, and enforcing organizational knowledge of desired security in the requirement base. Combining two requirement bases is called merging of requirements, and enforcement of organizational knowledge of information security to formally transform state of a requirement base is called harmonization of information security requirements.

This demonstrates three sources of information security requirements: upper layer requirement primitives, layer specific requirements, and unit specific requirement primitives. The upper layer requirements represent the security policy to be implemented by a specific unit, consisting of directives for coordinating formulation of requirements at that. Layer specific requirements are requirements common for each unit at a given layer and represent general organizational structuring of information security. Unit specific requirement primitives are preliminary requirements of the lower layer unit representing local adaptation of security need into a specific operational environment. Merging also consists of identification and resolving of potential conflicts there may be between various requirement primitives.

To enforce a hierarchy and to prevent unnecessary security design in units, a subset of units must be specified for each unit to illustrate the path of refinement in which requirements are refined through the organization. Two types of relationships between units at consecutive layers are *Child* relationship and *parent* relationship. Let U_n be a set of all units at layer n, and U_{n+1} a set of all units at layer $n+1$. Let $u \in U_n$ and $u' \in U_{n+1}$ be units. If exists a specification $(u, u') \in Child$ then u' is a *Child* unit of u. This means, requirement u enforces its requirement base into the requirement base of u' whenever merging of requirements occurs. Similarly, if exists a relationship $(u', u) \in Parent$ then u' receives requirement primitives according to which its requirement base must be modified from unit u.

Despite being simple, this way of specifying the organization in which information security requirements are dealt with has the advantage of allowing a simple notation for expressing the requirements. This requires extensive modeling of organization but since the modeling of organization does not need to be complete until abstract requirements are specified, the modeling of the organization can be a continuous process throughout the specification of information security requirements.

3 Expression of Information Security Requirements

The notation for expressing an information security requirement borrows the basic ideas from the theory of communicating sequential processes [13]. It is assumed, that exists an association A for sharing data between two processes P and Q. The existence of an association between two processes is the source of potential security violations, whether the communication takes place through an internal bus or a wide area public network. Communication of data over an association A always takes a form of a protocol p governing the content and sequence of messages communicated between P and Q. Each association and process has two attribute vectors, a vector of possession attributes and a vector of criteria attributes. Let R be a process or association, we denote possession attributes of R as $R.\phi$ and criteria attributes as $R.\chi$. Possession attributes, from security point of view, are mostly concerned with data coordinating security enforcement, such as cryptographic keys. Possession attributes set characteristics of possession attributes, for example length of keys.

Communication over association A, taking form of protocol p, must be protected by security enforcement algorithm a by applying possession attributes of processes and association. Further, the algorithm itself has a number of parameters, such as initial vectors for stream ciphers and block sizes for block ciphers. These are expressed similarly using notations $a.\phi$ and $a.\chi$. Each parameter has a name which uniquely identifies the parameter. This simplifies the notation, since there is no need to identify into which component a parameter is associated to. Therefore, each requirement can be expressed as a statement of form

$$(A; P; Q; p; a; pv) \tag{1}$$

where A, P, Q, p, and a are as above, and pv is a parameter vector of form pn op val where pn is a name of an attribute, op is an operator $=$, $\neq$, $<$, $\leq$, $>$, $\geq$, or $\in$ depending on the nature of pn. val is the value into which the value of attribute pn is compared to as specified by op.

It is likely that in practical applications, parameter vectors are only concerned with attribute vectors describing the security attributes instead of specifying fixed values for arguments. For theoretical considerations, though, it is essential the possession attributes are considered in the specification of security requirements.

A collection of requirements, expressed as in 1, is called a requirement base $c.rb$ where c is a layer or unit in an organization and $rb = \{r\}$ is a set of requirements r. Each layer and unit in an information security development organization has a separate requirement base, and transformations in requirement bases are done through four operations:

Addition of a requirement r to $c.rb$, as a result which $c.rb = c.rb \cup r$.
Removal of a requirement r from $c.rb$, as a result of which $c.rb = c.rb - r$.
Merging of requirements, that is combining two requirement bases through a specific merging functions.

Harmonization of requirements, where a subset of requirements in a requirement base are modified according to a specific harmonization function.

For the purposes of this paper, it is enough to assume existence of merging functions for combining requirement bases with simultaneous resolution of potential conflicts, and expressing and enforcing harmonization functions that modify the requirement bases that belong to the scope of harmonization. The purpose of harmonization is to increase consistency of requirements by enforcing a criteria that each requirement in the requirement scope must satisfy and to add detail during the derivation of technical security requirements from abstract security policy objectives. Merging is used for enforcing layer specific requirements into each unit at that layer and for transforming abstract requirement bases to lower levels of abstraction.

4 Grouping of Requirements

The mechanism for expressing information security development organization and information security requirements as atomic constructs enables strong functional grouping of requirements. That has two major advantages in the management of information security. First, requirement bases remain internally cohesive each one dealing with only one types of security enforcement. Second, the sizes of requirement bases remain smaller, leading to a more easily manageable sized subsets of requirements.

Assume an example organization illustrated in Fig. 2. There is an organizational unit responsible for secure communication at layer L_n, and this responsibility is delegated to a number of units at layer L_{n+1} according to the security service. For simplicity and intuitivity of the example, assume that there is only one association *Internet* of concern, connecting two systems P_1 and P_2, considered as processes. At the managerial layer L_n it is adequate to only express the intent on security without further details. Requirement

$$Internet; P_1; P_2, HTML; SSL; \emptyset \tag{2}$$

can be formulated to imply a managerial policy that communication over the Internet between the two processes that takes the form of HTML must be protected by SSL [11] security protocol with unspecified parameter vector. This is an adequate expression of intent for security, but not yet detailed enough to implement the actual security enforcement. Therefore, the requirement must be merged to units at layer L_{n+1} responsible for adding implementation details to the upper layer policy.

Since there are no layer specific requirements for layer L_{n+1} and requirement bases of each unit are initially empty, merging is a simple copying of each requirement in the requirement base of *Parent* unit to requirement bases of each *Child* unit. Detail can be added at each unit at layer L_{n+1} to differentiate the requirement base according to dedicated functionality. For example, unit dedicated

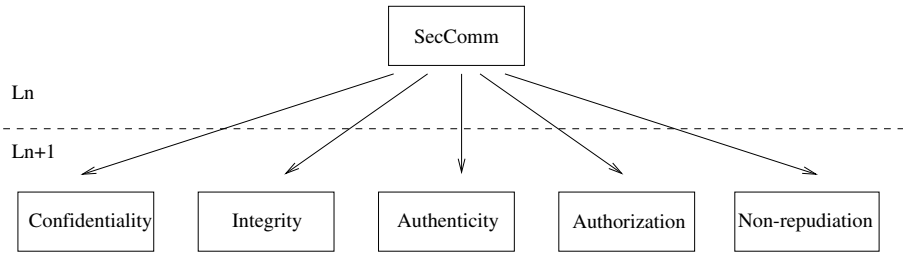


Fig. 2. Grouping of requirements

on confidentiality can focus on specification of desired encryption algorithms for SSL and required key lengths, constructing for example requirement

$$Internet; P_1; P_2; HTML; SSL; cs = DES; kl = 56 \tag{3}$$

stating that DES will be used as a cryptosystem for protocol payload with key length of 56 bits. *Authenticity* unit can focus on specification of digital signature algorithms and their key lengths and so on, for example

$$Internet; P_1; P_2; HTML; SSL; pkcs = RSA; hash = SHA, RSAkl = 1024 \tag{4}$$

to indicate that RSA with 1024 bit key will be used for authentication and SHA for secure hashing. All this can happen in parallel. Hence, security design has been decomposed into small components with only small amount of inter-dependencies and resources can be directed to parallel tasks and the time from design to implementation can be reduced, and it can be believed that since every responsibility for security is concerned with a small subset of all security functions, the likelihood of errors in design is reduced through easier dealing with requirements.

The problem with the approach is, that many security mechanism depend on similar parameters, such as cryptographic keys to achieve confidentiality, authenticity, integrity and on some extent non-repudiation. To reduce the number of duplicated tasks, an additional layer must be added to express dependencies between requirements and to establish a single point of responsibility for dealing with cryptographic keys.

5 Requirement Dependencies

The obvious approach towards requirement dependencies is to specialize each unit in the actual responsibility of enforcing that functionality and a number of related responsibilities. Once each unit does this, for example as illustrated in Fig. 3, functionalities such as key management can be separated from the security

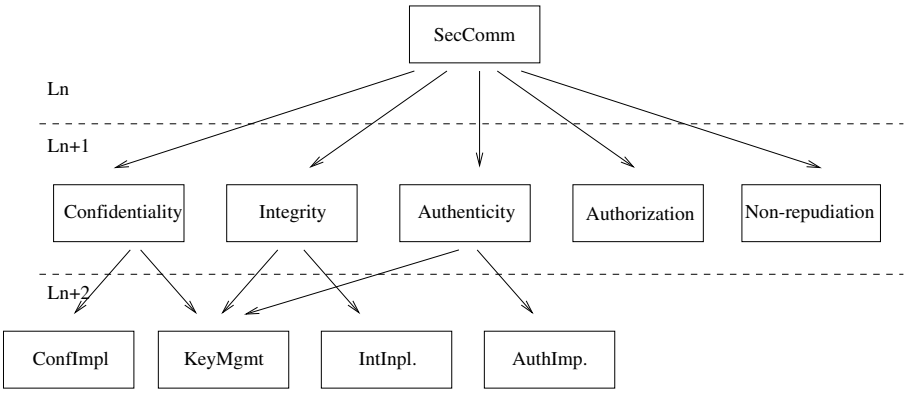


Fig. 3. Dependencies of requirements

functionality enforcement and responsibilities common to multiple units can be centralized under one administrative domain.

Units may make alterations on the structure of the system being modeled. For example, for confidentiality and authenticity processes P_1 and P_2 are adequate, but a *Authorization* unit would have a different view. One could require a mechanism to isolate processes P_1 and P_2 from direct communication by a firewall. Therefore, the local requirement base of unit *Authorization* would consist of process P_1 communication to the process FW through a LAN association, and process FW communication to process P_2 through the Internet association with separate security specification. For *Authorization* unit system P_2 appears either as one system or can be decomposed into filtering functionality and the actual system depending on the organizational business responsibility specifying ownership of P_2.

Authorizations introduce some limitations on the proposed notation. Only functional requirements can be expressed, operational considerations are out of the scope. System level authorizations can be expressed as described above, but file and object level authorizations introduce difficulties. Security algorithm in authorization can be expressed as a specific role based access control model, for example, but specification of a role access control policy and assignment of users to roles is merely an operational rather than functional security issue. Since role assignment is dynamic, it should not be expressed as a static requirement. Role structure and role authorizations are more static but may be subject to dynamic variation in time. Therefore, it is not clear whether they should be expressed dynamically or statically. Initial role structure and authorizations can be expressed statically in the initial design phase, but dynamics should be taken into account in operational considerations of systems developed.

When merging requirements, units at layer L_{n+1} transfer their refined requirements into both *Child* units and can then apply harmonization functions to further specialize them to match specific needs, whether enforcement of actual

functionality or dealing with defendant requirements. These further refinements in organizational structure and requirement descriptions are invisible to *parent* units. Therefore, requirements of an upper layer unit and all the related requirements can be enumerated to provide a comprehensive set of requirements directly and indirectly under that administrative domain. This enables monitoring the evolution of requirements and specification of corrective actions by those units coordinating many lower layer units.

6 Findings of a Case Study

A case study, fully documented in [17], has been conducted where the harmonization of information security requirements is applied in the medical informatics context. The core idea is to specify information security requirements for a system designed for sharing patient data among hospital and medical practitioners on a larger geographical Peninsula region, approximately 50 km east of Melbourne in Victoria, Australia. A number of functional components of the becoming system has been identified, each under different administrative domains. Yet, the coordination of design and implementation is centralized. Findings of applying the theory suggest that the main challenge in the specification of security requirements with a minimum amount of duplication, hence improved cost-efficiency, can be achieved through the organizational modeling of information security.

Initially, the delegation of the central responsibility and authority follows the administrative boundaries of systems. However, it quickly becomes the case when the actual security enforcement functions are to be established, that most components of the system share similar security functionality, mostly access control, authentication and encryption. Therefore, each of them can be best dealt with under a shared responsibility. Especially, since components under different administrative domains must cooperate, there is a risk of security violations unless a central authority is established to coordinate security of communication that exceeds administrative boundaries. Consequently, the initial delegation of responsibility appears to be spreading, suggesting that extensive modeling will be required. However, once the security service -level specifications are concluded, and corresponding security mechanisms are specified, quite a small number of security mechanisms are required to implement both direct and supporting security requirements.

Most security mechanisms provide a logical ending stage for the organizational modeling. Because of some higher level system design decisions, dedicated security enforcement software had to be used. Availability of (hopefully) tested and properly designed security enforcement components further limits the scope of organizational modeling, and also simplifies the specification of the actual security functions. However, the simplified design of security enforcement may lead to inefficiencies in the operation of the security enforcement subsystem. As each security software implements their own trust model and introduces different potential vulnerabilities, maintenance and operation of security, as well evalua-

tion, becomes more complicated. Yet, the conflict between the benefits achieved through the use of standardized security services and the potentially reduced efficiency of security enforcement and operation, together with the potential measures for efficiency, remains an area of further research.

7 Conclusions and Future Work

This paper has demonstrated theoretical considerations in the transformation of modeling effort in information security design from requirement modeling to organizational modeling. Many security development frameworks highlight the importance of organizational considerations but fail to deliver specifications of such models. The approach adopted enables decomposition of information security requirements into functional sub-components reducing the size of requirement bases, and expression of dependencies between requirements through organizational modeling. This keeps the size of requirement bases small and syntax for expressing information security requirements simple. These two factors are believed to contribute to the efficiency of the design of information security.

The notation for expressing requirements is only capable of capturing functional security requirements. Operational requirements are out of the scope. This does slightly limit the applicability in the specification of file and object level access control requirements. Since functional security requirements are expressed as static properties of an organizational component they are assigned to, high level of detail in access control requirements is difficult to achieve. There are a number of operational considerations associated to authorizations, such as specifying user-role assignments. As the major advantage of role based access control is the improved dynamics of this association, static definitions of roles would undermine the efficiency of role based access control.

It could be argued that the case is same with cryptographic keys but there is a significant difference of the random nature of many cryptographic keys. Public and private key pairs are considerable static, yet changing over time, but especially session keys established and exchanged in a session establishment phase should be very random. Therefore, a mechanism can be specified as expressed to generate such keys with no need to specify the content, whereas specification of user-role associations requires human intervention.

In addition to the relationship between the simplicity of design and efficiency of enforcement of security, additional further research could be carried on the integration of operational considerations, such as the role-user assignment, into the general model. Dynamic constructs used for actual execution of mechanisms to satisfy specific requirements are intentionally left outside the scope of the framework. Strong hierarchy and information hiding characteristics suggest that further integration of the framework to general system design could be achieved through object oriented modeling and would be a valuable research question to be addressed in the future.

References

1. A. Anderson, D. Longley, and L. F. Kwok. Security modeling for organizations. In *2nd ACM Conference on Computer and Communications Security*, pages 241–250, Fairfax, VA, USA, 1994. ACM Press. 248
2. J. Backhouse and G. Dhillon. Structures of responsibility and security of information systems. *European Journal of Information Systems*, 5(1):2–9, 1996. 249
3. D. Bailey. A philosophy of security management. In M. D. Abrams, S. Jajodia, and H. J. Podell, editors, *Information Security, An Integrated Collection of Essays*, pages 98–110. IEEE Computer Society Press, Los Alamitos, CA, USA, 1995. 248
4. R. Baskerville. Risk analysis as a source of professional knowledge. *Computers & Security*, 10(8), 1991. 249
5. R. Baskerville. Information systems security design methods: Implications for information systems development. *ACM Computing Surveys*, 25(4):375–414, December 1993. 249
6. D. E. Bell and L. LaPadula. Secure computer systems: Mathematical foundations and model. Technical Report M74-244, MITRE Corporation, Bedford, MA, USA, 1973. 247
7. Code of practise for information security management. British Standards Institute Standard BS 7799, UK, 1995. 249
8. International standard ISO/IEC 15408 common criteria for information technology security evaluation (parts 1-3), version 2.0, CCIB-98-026, May 1998. 248
9. R. Fisher. *Information Systems Security*. Prentice-Hall, 1984. 249
10. W. Ford. *Computer Communications Security: Principles, Standard Protocols, and Techniques*. Prentice Hall, Inc., Englewood Cliffs, NJ, USA, 1995. 248
11. A. O. Freier, P. Karlton, and P. C. Kocher. The SSL protocol, version 3.0. Internet-draft draft-freier-ssl-version3-02.txt, November 18 1996. 254
12. J. A. Goguen and J. Mesegeur. Unwinding and inference control. In *Proceedings of the 1984 IEEE Symposium on Security and Privacy*, 1984. 247
13. C. A. R. Hoare. *Communicating Sequential Processes*. Prentice Hall, London, UK, 1985. 253
14. IT baseline protection manual. BSI, Germany, 1996. 249
15. B. Lampson. Protection. *ACM Operating Systems review*, 8:18–24, 1974. 247
16. L. J. LaPadula. Foreword for republishing of the Bell-LaPadula model. *Journal of Computer Security*, 4:233–238, 1996. 248
17. J. Leiwo. *Harmonization of Information Security Requirements*. PhD thesis, Monash University, 1999. 250, 257
18. J. Leiwo and Y. Zheng. A formal model to aid documenting and harmonizing of information security requirements. In L. Yngström and J. Carlsen, editors, *Information Security in Research and Business, Proceedings of the IFIP TC11 13th International Conference on Information Security (SEC'97)*, pages 25–38. Chapman & Hall, May 14 – 16 1997. 250, 251
19. J. Leiwo and Y. Zheng. A framework for the management of information security requirements. In *Information Security, Proceedings of the First International Workshop*, number 1396 in Lecture Notes in Computer Science, pages 232–245. Springer–Verlag, 1997. 250
20. D. McCullough. Specifications for multi-level security and hook-up property. In *Proceedings of the 1987 IEEE Symposium on Security and Privacy*, pages 161–166, 1987. 247

21. D. B. Parker. *Managers Guide to Computer Security.* Prentice-Hall, Inc, Reston, VA, USA, 1981. 249
22. C. Salter, O. S. Saydjari, B. Schneier, and J. Wallner. Toward a secure system engineering methodology. In *Proceedings of the New Security Paradigms Workshop.* ACM Press, September 1998. 249
23. R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman. Role-based access control models. *IEEE Computer*, pages 38–47, February 1996. 248
24. S. Smith. LAVA's dynamic threat analysis. In *Proceedings of the 12th National Computer Security Conference*, 1989. 249
25. D. F. Sterne. On the buzzwork "security policy". In *Proceedings of the 1991 IEEE Symposium on Research in Security and Privacy*, pages 219–230. IEEE Computer Society Press, 1991. 248
26. D. Sutherland. A model of information. In *Proceedings of the 9th National Computer Security Conference*, 1986. 247
27. R. von Solms. Information security management: The second generation. *Computers & Security*, 15(4):281–288, 1996. 249
28. J. D. Weiss. A system security engineering process. In *Proceedings of the 14th National Computer Security Conference*, 1991. 249

A Closed Approach to Vague Collections in Partly Inaccessible Distributed Databases

Oliver Haase[1] and Andreas Henrich[2]

[1] NEC Europe Ltd, C&C Research Laboratories Heidelberg,
Adenauerplatz 6, D-69115 Heidelberg, Germany
Oliver.Haase@ccrle.nec.de
[2] Otto-Friedrich-Universität Bamberg, Fakultät Sozial- und Wirtschaftswissenschaften,
Praktische Informatik, D-96045 Bamberg, Germany
Andreas.Henrich@sowi.uni-bamberg.de

Abstract. Inaccessibility of part of the database is a research topic hardly addressed in the literature about distributed databases. However, there are various application areas, where the database has to remain operable even if part of the database is inaccessible. In particular, queries to the database should be processed in an appropriate way. This means that the final and all intermediate results of a query in a distributed database system must be regarded as potentially vague.
In this paper we propose hybrid representations for vague sets, vague multisets, and vague lists, which address the accessible elements in an enumerating part and the inaccessible elements in a descriptive part. These representations allow to redefine the usual query language operations for vague sets, vague multisets, and vague lists. The main advantage of our approach is that the descriptive part of the representation can be used to improve the enumerating part during query processing.

1 Addressed Problem

An interesting problem in the field of distributed database systems is query evaluation when a part of the database is temporarily inaccessible. If a database is distributed on top of an unreliable communication basis, e.g. the Internet, a network or machine failure can cause such a situation. Similar conditions may occur during the intended use of the system, if part of the database is stored on portable workstations, which can be disconnected from the network arbitrarily. Such situations are e.g. considered in the distribution model of PCTE [6], the ISO standard for an open repository, and related situations have to be handled in mobile environments as described in [11].

For application areas like those mentioned above it is desirable that the accessible part of the database still remains operable. In particular queries to the database should be processed in an appropriate way. Of course, there are applications that cannot do without a complete, one hundred percent correct result. These applications must be built on a reliable network, they must use

Eder et al. (Eds.): ADBIS'99, LNCS 1691, pp. 261–274, 1999.
© Springer-Verlag Berlin Heidelberg 1999

additional techniques like replication, and last but not least they must stop query processing when the database is not completely accessible and wait for a recovered situation instead. On the other hand, there are applications for which an approximation of the result is better than no result at all.

Unfortunately, in these circumstances it is not sufficient to process queries in the usual way, and to handle inaccessibility by taking only the accessible part of the database into account. Instead there are two sources causing the necessity of adapted query processing techniques:

The first reason is that in the case of partial inaccessibility the usual two-valued logic does not meet our requirements: If the evaluation of a selection criterion needs access to inaccessible data, it cannot yield one of the truth values *true* or *false*, but has to deliver an *unknown* result. The use of a three-valued logic in turn implies the use of *vague result collections* instead of *crisp result collections*. That is due to the elements that might belong to the result but of which it is not sure if they really do.

The second reason concerns the incompleteness of the computed result: In principle all accessible sure elements of the result as well as all accessible uncertain elements of the result can be computed. To this end the corresponding query must be transformed to a so-called *query condition*. The query condition is a predicate that can be applied to a potential candidate for the result; it will evaluate to *true*, if the candidate belongs to the result, it will evaluate to *false*, if the candidate does not belong to the result, and it will evaluate to *unknown*, if the candidate may or may not belong to the result. In an absolute declarative query language, e.g. the relational calculus, queries are already formulated as query conditions. In the case of complete accessibility of the database, the result consists of all elements fulfilling the query condition.

If the database is partly inaccessible, one possible query evaluation is to check each accessible element against the corresponding query condition. By doing so, the (vague) result consists of all accessible elements that can be or really are in the desired crisp result. However, this *predicative query evaluation* is very costly, and therefore normally query evaluation is not predicative but *constructive*; i.e. the evaluation commences with some particular start elements – e.g. the extension of a certain object type – and navigates successively to the desired result elements. These navigations can involve logical relationships (e.g. formulated by means of a theta join) as well as physical relationships (e.g. relationships in an E-R-data model). If a query is evaluated constructively, partial inaccessibility may break some navigation paths from certain start elements to the corresponding result elements. These breaks can be caused by the inaccessibility of intermediate elements. Thus, a result element can be **unreachable** with respect to the execution plan whilst still being **accessible**.

Hence, constructive query evaluation in partly inaccessible databases leads to *incomplete* vague result collections. This incompleteness is a big drawback, both for the end-user and the query processing itself. For the end-user it means the system answering as follows: there are some elements fulfilling the query; additionally there are some elements *maybe* fulfilling your query; and finally each

element of the database is also a potential candidate for your query, since the computed result is incomplete. Evidently, this answer is far from being satisfying.

Incompleteness is also a problem during the query processing: Assume we have the vague results of two sub-queries, each of which being incomplete due to partial inaccessibility. If these vague sets are to be combined by a binary operation, each of both must regard the other one as consisting of potential candidates for itself. Hence, all elements occurring in at least one vague input collection must be taken into account for the vague result collection. This uncertainty leads to a growing vagueness of the final result during the query processing. Much of that vagueness is not forced by the partial inaccessibility, but by an awkward query processing. The predicative evaluation of the same query would lead to immensely less vagueness in the result. What we need is a technique that combines the low costs of a constructive evaluation with the accuracy of a predicative query evaluation.

To sum up, the situation is as follows: Partial inaccessibility of the database leads to a certain vagueness of the result. The intuitive representation of this kind of vagueness is to distinguish between the sure and the uncertain part of the result. But due to the normal constructive evaluation of a query, vague collections are in general incomplete. Thus, a representation as well as adapted operations are needed that overcome the described problems.

2 Related Work

In the area of *relational databases* much work has been done to deal with unknown or incomplete data. The basic concept is *null values* [5,18,12,13,8].

In general null values are used to cope with the vagueness of the stored information — usually induced by unknown or undefined attribute values — which may result in the vagueness of a query result. However, a fundamental difference between null values and inaccessible parts of a database is the following: a null value stands for an unknown *single attribute*; the values of the remaining attributes and the existence of the whole object (tuple) are known. If parts of a database are inaccessible, even the *existence* of objects is unknown.

Another approach to describe vague values is the use of fuzzy sets. A prerequisite for this approach is probabilistic information, such as the distribution of the concerned attributes. Some relevant papers in this respect are [3,7,15].

The articles [1,14,19,20,16] are concerned with vague sets resulting from vague attribute values – sometimes called *rough sets*. To this end, the result is approximated by a lower bound (containing all objects surely contained in the desired crisp result) and an upper bound (containing all objects potentially contained in the desired result). The most important difference between these interpretations of vague sets and our approach is that all papers mentioned above presuppose the accessibility of all relevant data. Furthermore, the mentioned papers deal with sets only, whereas multisets and lists are not considered.

The work presented in [4,17,2] deals with partial inaccessibility as follows: in this case the query processor yields a so-called *partial result* that consists

of a partly evaluated query. This query contains the used accessible parts of the database materialized as constants. It can be evaluated at a later point of time, when the accessibility situation will have changed. This work is based upon the assumption that in an unreliable environment the partial inaccessibility takes only short time and that it concerns changing nodes. Only in these circumstances it is useful to materialise the accessible data; otherwise the pure redo of the query evaluation has the same effect. The facilities to extract relevant data from the partial results are rather low. To this end the user himself has to formulate so-called *parachute queries* that operate on the partial results. Another assumption of this approach is the relational data model with no horizontal segmentation of the relations. Hence, in the case of partial inaccessibility a relation is completely accessible or it is completely inaccessible; a restriction we do not impose.

In [9] we have presented a first approach to deal with vague *sets* in the sketched scenario and in [10] a comprehensive discussion of the mathematical foundations of our representation of vague sets is given together with the correctness proofs and a short outlook on multisets. In contrast the present paper adds convenient representations for vague *multisets* and vague *lists*. Hence, we propose a *closed* approach to deal with partial inaccessibility that covers *all three kinds of collections*. Furthermore we consider aggregate functions.

3 Example Environment

Assume an object-oriented or E-R-based distributed database management system supporting links to represent relationships between objects. Further assume a schema for modelling diagrams for object-oriented analysis (OOA) methods, described as follows: An object of type *OOA_Diagram* is connected with the classes defined in the diagram via links of type *contains* representing the whole-part-relationship between the diagram and its classes. Furthermore, two attributes (*name* and *comment*) are defined for the object type *OOA_Diagram*. Five originating link types are defined for the object type *Class*. Links of the types *has_attribute* and *has_method* represent the relationships to the attributes and methods of the class. Links of the types *has_subclass* and *has_superclass* define the position of the class in the inheritance graph. Furthermore, links of type *in* lead to the diagram containing the class definition. Finally, a method is connected with its parameters via links of type *has_parameter*.

Now, let us assume that the objects stored in the database are distributed on four segments as depicted in figure 1 and that *segment 3* is inaccessible at the moment. For the following examples it is necessary to define the physical location of the depicted links. To this end let us assume that they are stored together with their origin objects, i.e. a link originating from an object located on *segment 2* resides on *segment 2*, too.

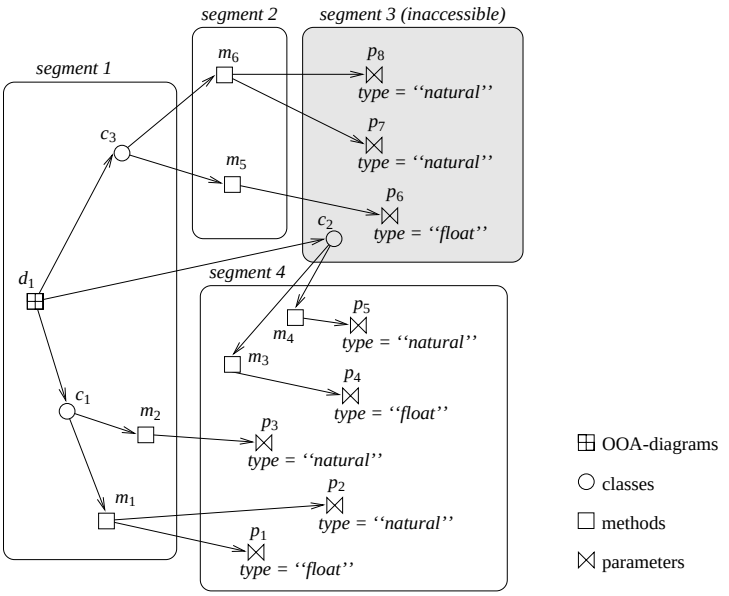


Fig. 1. An example database

4 Vague Sets

In the following we will shortly describe the representation of vague sets. For a more thorough description including illustrative examples please refer to [9,10].

A vague set V representing the result of a query processed on a partly inaccessible database can be envisaged as a set of sets. Each element is a set which could be the correct answer to the computed query, if the entire database was accessible. Informally speaking, a vague set V is defined by: (1) A set $\widehat{V}_\lambda$ that consists of the *accessible* and *reachable* elements which surely belong to the correct answer, (2) a set $\widehat{V}_\upsilon \supseteq \widehat{V}_\lambda$ that consists of the accessible and reachable elements for which we cannot be sure that they do *not* belong to the correct answer and (3) a three-valued logical predicate δ_V, the so-called descriptive component, that states for each element i of the foundation set, if it *surely* belongs to the correct answer ($\delta_V(i) \equiv t$), if it *maybe* belongs to the correct answer ($\delta_V(i) \equiv u$), or if it surely *does not* belong to the correct answer ($\delta_V(i) \equiv f$).

Formally, a vague set over the foundation set T can be defined as follows:

Definition 1 (vague sets). *The triple* $V = (\widehat{V}_\lambda, \widehat{V}_\upsilon, \delta_V)$ *with* $\widehat{V}_\lambda \subseteq T$, $\widehat{V}_\upsilon \subseteq T$, $\delta_V : T \to \{t, f, u\}$ *is a* vague set *over the foundation set* T *(*$V \in \widetilde{\mathbf{Set}}(T)$*), iff*

$$\widehat{V}_\lambda \subseteq \widehat{V}_\upsilon \quad \wedge \quad \forall i \in \widehat{V}_\lambda : \delta_V(i) \equiv t \quad \wedge \quad \forall i \in \widehat{V}_\upsilon \setminus \widehat{V}_\lambda : \delta_V(i) \equiv u$$

Furthermore, the sets V_λ *and* V_υ *are defined as* $V_\lambda = \{i \in T : \delta_V(i) \equiv t\}$ *and* $V_\upsilon = \{i \in T : \delta_V(i) \not\equiv f\}$. *A set* $S \subseteq T$ *is said to be contained in* V*, iff it is* between V_λ *and* V_υ*:* $S \in V \Leftrightarrow V_\lambda \subseteq S \subseteq V_\upsilon$. □

To show the use of this representation, we describe the adaptation of two query language operators to vague sets, namely set intersection and selection.

In the following we use the index "3", if we refer to operations on vague sets or to three-valued logical junctions, resp. If an operator or logical junction is written without index, we always refer to the exact or two-valued case, resp.

Let us consider the vague set intersection as an example for the usual set operations $\cup$, $\cap$, and $\setminus$. For $X = V \cap_3 W$, we define:

$$\begin{aligned} \widehat{X}_\lambda &= \{i \in \widehat{V}_\upsilon \cup \widehat{W}_\upsilon \mid \delta_V(i) \equiv t \wedge \delta_W(i) \equiv t\} \\ \widehat{X}_\upsilon &= \{i \in \widehat{V}_\upsilon \cup \widehat{W}_\upsilon \mid \delta_V(i) \not\equiv f \wedge \delta_W(i) \not\equiv f\} \\ \delta_X(i) &\equiv \delta_V(i) \vee_3 \delta_W(i) \end{aligned}$$

The rules for the vague set intersection demonstrate the basic principle for all binary set operations. We consider *all* explicitly known elements, i.e. all $i \in \widehat{V}_\upsilon \cup \widehat{W}_\upsilon$. This is identical to the proceeding if we only knew that V and W were incomplete. The difference is that we check the desired properties for each element by the help of the descriptive components in a second step.

Another important query language operator is the selection σ. Its adaptation to vague sets maps a vague set and a three-valued selection predicate to a vague result set: $\sigma_3 : \widetilde{\mathbf{Set}}(T) \times (T \to \{t, f, u\}) \to \widetilde{\mathbf{Set}}(T)$. The semantics is as follows:

$$\begin{aligned} \sigma_3(V, P_3) = X, \text{ with } \widehat{X}_\lambda &= \{i \in \widehat{V}_\lambda \mid P_3(i) \equiv t\} \\ \widehat{X}_\upsilon &= \{i \in \widehat{V}_\upsilon \mid P_3(i) \not\equiv f\} \\ \delta_X(i) &= \delta_V(i) \wedge_3 P_3(i) \end{aligned}$$

As an example for a relation on vague sets over T we state the vague set inclusion:

$$(V \subseteq_3 W) \equiv \begin{cases} t \text{ , if } \forall i \in T : (\delta_V(i) \Rightarrow_3 \delta_W(i)) \equiv t \\ f \text{ , if } \exists i \in T : (\delta_V(i) \Rightarrow_3 \delta_W(i)) \equiv f \\ u \text{ , otherwise} \end{cases}$$

5 Vague Multisets

Analogously to vague sets, a vague *multi*set is a set of *multi*sets where each element is a multiset which could be the correct answer to the computed query. Applying the base idea underlying vague *sets* – using a hybrid representation which comprises an explicit, enumerated part and a descriptive part – to *multisets*, we get

1. A set $\Delta_\mathcal{V} \subseteq T$, that comprises those potential elements of the multiset $\mathcal{V}$ we actually have under access,
2. a mapping $\lambda_\mathcal{V} : T \to \mathbb{N}_0$ that maps each element i of the foundation set to its minimum number of occurrences in $\mathcal{V}$, and
3. a mapping $\upsilon_\mathcal{V} : T \to \mathbb{N}_0^\infty$ that maps each element i of the foundation set to its maximum number of occurrences in $\mathcal{V}$; the notation $\mathbb{N}_0^\infty$ means the set $\mathbb{N}_0 \cup \{\infty\}$, i.e. the extension of $\mathbb{N}_0$ by a maximum element to a complete lattice.

Formally, a vague multiset can be defined as follows:

Definition 2 (vague multisets). *A vague multiset over the foundation set* T *(*$\mathcal{V} \in \widetilde{\mathbf{Bag}}(T)$*) is a triple* $\mathcal{V} = (\Delta_\mathcal{V}, \lambda_\mathcal{V}, \upsilon_\mathcal{V})$*, with the components* $\Delta_\mathcal{V} \subseteq T$*,* $\lambda_\mathcal{V} : T \to \boldsymbol{N}_0$*,* $\upsilon_\mathcal{V} : T \to \boldsymbol{N}_0^\infty$*, iff the following conditions hold:*

$$\forall i \in T : \lambda_\mathcal{V}(i) \leq \upsilon_\mathcal{V}(i) \quad \wedge \quad \forall i \in \Delta_\mathcal{V} : \upsilon_\mathcal{V}(i) > 0$$

The relation $\in$ *between a multiset* $\mathcal{A} \in \mathbf{Bag}(T)$ *and a vague multiset* $\mathcal{V} \in \widetilde{\mathbf{Bag}}(T)$ *is defined as follows:* $\mathcal{A} \in \mathcal{V} \Leftrightarrow \forall i \in T : \lambda_\mathcal{V}(i) \leq \mathcal{A}(i) \leq \upsilon_\mathcal{V}(i)$ □

The above definition ensures that we do not carry elements in $\Delta_\mathcal{V}$ which are surely not contained in the vague multiset $\mathcal{V}$.

Let us consider a simple example that shows the use of our representation:

Example 1. Assume the query: *Select the types of the parameters of all methods of all classes of the diagram called "Billing"!*, where the result should contain duplicates.

A natural way to evaluate this query would be to start from the diagram d_1 and to navigate via the appropriate links. In this case the component $\Delta_\mathcal{V}$ of the resulting vague multiset $\mathcal{V}$ would be $\Delta_\mathcal{V} = \{\mathit{natural}, \mathit{float}\}$, because we would only reach p_1, p_2, and p_3.

Let O denote the set containing all objects in the database. Further assume that the function $\mathit{val}(o, a)$ yields the value of the attribute a for the object o and that the three-valued logical predicate $\mathit{pred}_3 : O \to \{t, f, u\}$ requires (1) that a candidate must be of type *Parameter*, (2) that it must have an incoming link of type *has_parameter* with origin m, (3) that m must have an incoming link of type *has_method* with origin c, (4) that c must have an incoming link of type *contains* with origin d, and (5) that the value of the attribute *name* of d must be "Billing". Obviously each sub-predicate of pred_3 must be evaluated in a three-valued manner. Then we can define the functions $\lambda_\mathcal{V}$ and $\upsilon_\mathcal{V}$ as follows:

$$\lambda_\mathcal{V}(i) = \begin{cases} 1 \text{ , if } i = \mathit{float} \vee (\exists_3 p \in O : i = \mathit{val}(p, \mathit{type}) \wedge \mathit{pred}_3(p)) \equiv t \\ 2 \text{ , if } i = \mathit{natural} \\ 0 \text{ , otherwise} \end{cases}$$

$$\upsilon_\mathcal{V}(i) = \begin{cases} 0 \text{ , if } (\exists_3 p \in O : i = \mathit{val}(p, \mathit{type}) \wedge \mathit{pred}_3(p)) \equiv f \\ \infty \text{ , otherwise} \end{cases}$$

This definition takes into account that (1) two occurrences of *natural* and one occurrence of *float* are reached, and that (2) we know that there might be missing elements.

Now we can show the adaptation of two query language operators to vague multisets. First we consider the vague multiset intersection $\cap_{m,3}$ as an example for a typical base multiset operation [1]. To compute $\mathcal{X} = \mathcal{V} \cap_{m,3} \mathcal{W}$, we define:

$$\begin{aligned} \Delta_\mathcal{X} &= \{i \in \Delta_\mathcal{V} \cup_m \Delta_\mathcal{W} \mid \upsilon_\mathcal{X}(i) > 0\} \\ \lambda_\mathcal{X}(i) &= \min(\lambda_\mathcal{V}(i), \lambda_\mathcal{W}(i)) \\ \upsilon_\mathcal{X}(i) &= \min(\upsilon_\mathcal{V}(i), \upsilon_\mathcal{W}(i)) \end{aligned}$$

[1] Note that we use the index m to distinguish set and multiset operations.

The other operator we consider is the selection on multisets σ_m. Its adaptation to vague multisets maps a vague multiset and a three-valued selection predicate to a vague result multiset: $\sigma_{m,3} : \widetilde{\mathbf{Bag}}(T) \times (T \to \{t, f, u\}) \to \widetilde{\mathbf{Bag}}(T)$ The semantics is as follows:

$$\sigma_{m,3}(\mathcal{V}, P_3) = \mathcal{X}, \text{ with } \Delta_\mathcal{X} = \{i \in \Delta_\mathcal{V} \mid P_3(i) \not\equiv f\}$$

$$\lambda_\mathcal{X}(i) = \begin{cases} \lambda_\mathcal{V}(i) \text{ , if } P_3(i) \equiv t \\ 0 \quad \text{, otherwise} \end{cases}$$

$$\upsilon_\mathcal{X}(i) = \begin{cases} \upsilon_\mathcal{V}(i) \text{ , if } P_3(i) \not\equiv f \\ 0 \quad \text{, otherwise} \end{cases}$$

As an example for a relation on vague multisets over T we state the vague multiset inclusion:

$$(\mathcal{V} \subseteq_{m,3} \mathcal{W}) \equiv \begin{cases} t \text{ , if } \forall i \in T : \upsilon_\mathcal{V}(i) \leq \lambda_\mathcal{W}(i) \\ f \text{ , if } \exists i \in T : \lambda_\mathcal{V}(i) > \upsilon_\mathcal{W}(i) \\ u \text{ , otherwise} \end{cases}$$

6 Lists

Before we actually deal with *vague* lists, we explain our understanding of an *exact* list in the context of query languages.

6.1 Exact Lists

In the context of query languages, a list over the foundation set T is usually envisaged as a mapping $\mathbb{N} \to T$. In our opinion this is not exactly what we need, because in most cases a list arises from sorting a (multi)set. The sorting criteria actually used to order a multiset (1) build a partition of the multiset where each subset (= rank) consists of elements with the same value for the sorting criterion, and (2) impose an irreflexive order upon these ranks.

To become more precise, we give an example: Assume a set of employees which is to be sorted with respect to their ages. There may be different employees with the same age. If we want to represent the result list of the sorting operation as a mapping $\mathbb{N} \to \mathbf{employee}$, we have to impose an additional order on the employees of the same age. By doing so, we lose a certain degree of freedom, that we conceptually could have. Of course, the representation of a query result usually assumes the additional order, too, but its early introduction during query processing is not necessary.

Another approach would be to represent a list by a multiset together with an irreflexive order over the different elements. But unfortunately — e.g. due to list concatenation — one and the same element of the foundation set can happen to occur at *disconnected* places in a list. To handle these cases, we identify each separated cluster of the same element i by a unique number n, and we represent an exact list by a multiset where the individual pairs (i, n) constitute the domain, together with an irreflexive order over these pairs.

Definition 3 (exact lists). *An exact list* $\mathtt{A}$ *over the foundation set* T *(*$\mathtt{A} \in \mathbf{List}(T)$*) is a pair* $(o_{\mathtt{A}}, <_{\mathtt{A}})$*, with* $o_{\mathtt{A}} : T \times \boldsymbol{N} \to \boldsymbol{N}_0$*,* $<_{\mathtt{A}} \subseteq (T \times \boldsymbol{N}) \times (T \times \boldsymbol{N})$*. The components are defined as follows:*

1. $o_{\mathtt{A}}$ *is a multiset over* $(T \times \boldsymbol{N})$*, i.e.* $o_{\mathtt{A}} \in \mathbf{Bag}(T \times \boldsymbol{N})$*.*
2. $<_{\mathtt{A}}$ *is an irreflexive order over the relevant part* $\rho(\mathtt{A}) = \{(i,n) \in (T \times \boldsymbol{N}) \mid o_{\mathtt{A}}(i,n) > 0\}$ *of* $\mathtt{A}$*, i.e. it is (a) irreflexive:* $\forall x \in \rho(\mathtt{A}) : x \not<_{\mathtt{A}} x$*, (b) transitive:* $\forall x,y,z \in \rho(\mathtt{A}) : x <_{\mathtt{A}} y \wedge y <_{\mathtt{A}} z \Rightarrow x <_{\mathtt{A}} z$*, and (c) trichotome:* $\forall x,y \in \rho(\mathtt{A}) :$ *exactly one of the relations* $x <_{\mathtt{A}} y$*,* $y <_{\mathtt{A}} x$*, or* $x =_{\mathtt{A}} y$ *is true (the equality '*$=_{\mathtt{A}}$*' is interpreted as the equivalence of two elements relative to* $<_{\mathtt{A}}$*).* □

Example 2. Assume the query: *Calculate a list of all methods of all classes in the diagram named "Billing" which have at least one natural parameter, sorted with respect to the number of their natural parameters!*, evaluated on the database in figure 1, but *segment 3* **being accessible**.
The resulting list $\mathtt{A}$ could be given by

$$o_{\mathtt{A}}(i) = \begin{cases} 1 \text{ , if } i \in \{(m_1,1),(m_2,1),(m_6,1)\} \\ 0 \text{ , otherwise} \end{cases}$$
$$<_{\mathtt{A}} \; : \; (m_1,1) =_{\mathtt{A}} (m_2,1) <_{\mathtt{A}} (m_6,1)$$

There are two points to be mentioned with respect to this example: (1) Because the list arises from a set instead of a multiset, both the mapping $o_{\mathtt{A}}$ and the numbering of the individual "clusters" of the same element (that have cardinality 1) are actually unnecessary. (2) We numbered the elements of $\rho(\mathtt{A})$ only for reasons of simplicity by 1. The numbering can be arbitrary.

From a mathematical point of view we first build the partition of $\rho(\mathtt{A})$ and then we impose an order on this partition. Nevertheless — from a technical point of view — both steps can be computed simultaneously. To this end, we evaluate the predicate $x <_{\mathtt{A}} y$ as well as $y <_{\mathtt{A}} x$. This implicitly yields the partition of $\rho(\mathtt{A})$, because those pairs x, y for which neither $x <_{\mathtt{A}} y$ nor $y <_{\mathtt{A}} x$ holds belong to the same rank.

6.2 Vague Lists

If we take inaccessibility into account, not only the elements of a list may become uncertain, but also the ordering may become vague, e.g. because the complete evaluation of the order criterion would require access to inaccessible parts of the database. When we evaluate the ordering $<_3$ three-valued, the results given in table 1 are possible [2]. The missing combinations are not possible due to the trichotomy of an irreflexive order.

Definition 4 (vague order). *The three-valued logical predicate* $<_3 : U \times U \to \{t,f,u\}$ *is a vague order over* U*, iff*

[2] The question mark means that we have no information about the relation between x and y.

$x <_3 y$	f	f	f	u	u	t
$y <_3 x$	f	u	t	f	u	f
relation between x and y	$=$	$\geq$	$>$	$\leq$	?	$<$

Table 1. Relation between x and y for $<_3$

1. $\forall x \in U : (x <_3 x) \equiv f$ *(irreflexivity)*
2. $\forall x, y \in U : (x <_3 y) \equiv t \Rightarrow (y <_3 x) \equiv f$ *(trichotomy)*
3. $\forall x, y, z \in U : (y <_3 x) \equiv f \wedge (z <_3 y) \equiv f$
   $\Rightarrow (z <_3 x) \equiv f \wedge (x <_3 z) \equiv (x <_3 y) \vee_3 (y <_3 z)$ *(transitivity)*

An exact order $<$ *over* U *is said to be contained in* $<_3$ *(*$<\in<_3$*), iff* $\forall x, y \in U : x < y \Rightarrow (x <_3 y) \not\equiv f$. □

If we interpret condition 3 (transitivity) by means of table 1, it can be read as: $\forall x, y, z \in T : x \leq y \wedge y \leq z \Rightarrow x \leq z \wedge (x < z \Leftrightarrow x < y \vee y < z)$

Now that we have defined a vague order, we can also define a vague list $\mathtt{V} \in \widetilde{\mathbf{List}}(T)$ as a quadruple consisting of three components that constitute a vague multiset over $T \times \boldsymbol{N}$ (as mentioned before, the bundling of an element of T with a number serves for the numbering of the individual clusters of elements of T), and one component that builds a vague order over the vague multiset.

Definition 5 (vague lists). *A vague list over the foundation set* T *(*$V \in \widetilde{\mathbf{List}}(T)$*) is a quadruple* $\mathtt{V} = (\Delta_\mathtt{V}, \lambda_\mathtt{V}, \upsilon_\mathtt{V}, \prec_\mathtt{V})$*, with the components* $\Delta_\mathtt{V} \subseteq T \times \boldsymbol{N}$*,* $\lambda_\mathtt{V} : T \times \boldsymbol{N} \to \boldsymbol{N}_0$*,* $\upsilon_\mathtt{V} : T \times \boldsymbol{N} \to \boldsymbol{N}_0^\infty$*,* $\prec_\mathtt{V}$*:* $(T \times \boldsymbol{N}) \times (T \times \boldsymbol{N}) \to \{t, f, u\}$*, iff the following conditions hold:*

1. $(\Delta_\mathtt{V}, \lambda_\mathtt{V}, \upsilon_\mathtt{V}) \in \widetilde{\mathbf{Bag}}(T \times \boldsymbol{N})$.
2. $\prec_\mathtt{V}$ *is a vague order over the relevant part* $\rho(\mathtt{V}) = \{(i, n) \in (T \times \boldsymbol{N}) \mid \upsilon_\mathtt{V}((i, n)) > 0\}$ *of* $\mathtt{V}$.

The relation $\in$ *between a list* $\mathtt{A} \in \mathbf{List}(T)$ *and a vague list* $\mathtt{V} \in \widetilde{\mathbf{List}}(T)$ *is defined as:* $\mathtt{A} \in \mathtt{V} \Leftrightarrow (o_\mathtt{A} \in (\Delta_\mathtt{V}, \lambda_\mathtt{V}, \upsilon_\mathtt{V}) \wedge <_\mathtt{A} \in \prec_\mathtt{V})$. □

Example 3. On the database presented in figure 1 we evaluate the following query: "*Calculate a list of all methods of all classes in the diagram named "Billing" which have at least one float parameter, sorted with respect to the number of their float parameters!*"

To state the vague result list of this query we use (1) an auxiliary predicate $pred_3 : O \to \{t, f, u\}$ which requires that a candidate must be a method of a class of the diagram named "Billing" having at least one float parameter, and (2) the function $func_3 : O \to \mathcal{P}(\boldsymbol{N}_0)$ which determines the vague number of float parameters of a method. Then the vague result list can be given as:

$$\begin{aligned}
\Delta_{\mathtt{V}} &= \{(m_1,1),(m_5,1),(m_6,1)\}\\
\lambda_{\mathtt{V}}((i,n)) &= \begin{cases} 1 \text{ , if } (i,n)=(m_1,1) \vee \mathit{pred}_3(i) \equiv t \wedge n=1\\ 0 \text{ , otherwise}\end{cases}\\
\upsilon_{\mathtt{V}}((i,n)) &= \begin{cases} 0 \text{ , if } \mathit{pred}_3(i) \equiv f \vee n=0\\ 1 \text{ , otherwise}\end{cases}\\
\prec_{\mathtt{V}}((i,n),(j,m)) &= \begin{cases} f \text{ , if } n=m=1 \wedge (i,j)=(m_1,m_5)\\ u \text{ , if } n=m=1 \wedge (i,j) \in \{(m_1,m_6),\\ \qquad (m_5,m_6),(m_5,m_1),(m_6,m_1),(m_6,m_5)\}\\ \mathit{func}_3(i) <_3 \mathit{func}_3(j), \text{otherwise}\end{cases}
\end{aligned}$$

6.3 Vague List Operations

Now we explain the adaptation of list concatenation and selection to vague lists.

Whenever we combine two lists $\mathtt{V}$ and $\mathtt{W}$, we always presuppose (due to technical reasons) that the relevant elements $i \in \rho(\mathtt{V})$ and $j \in \rho(\mathtt{W})$ are numbered differently, i.e. an element (i,n) is never contained both in $\rho(\mathtt{V})$ and in $\rho(\mathtt{W})$. This premise can easily be achieved by renumbering the corresponding elements.

As an example for a base operation on vague lists we show the adaptation of the list concatenation $\circ$ to vague lists. In order to compute $\mathtt{X} = \mathtt{V} \circ_3 \mathtt{W}$, we define:

$$\begin{aligned}
\Delta_{\mathtt{X}} &= \Delta_{\mathtt{V}} \cup \Delta_{\mathtt{W}}\\
\lambda_{\mathtt{X}}(k) &= \begin{cases} \lambda_{\mathtt{V}}(k) \text{ , if } k \in \rho(\mathtt{V})\\ \lambda_{\mathtt{W}}(k) \text{ , if } k \in \rho(\mathtt{W})\\ 0 \quad \text{ , otherwise}\end{cases}\\
\upsilon_{\mathtt{X}}(k) &= \begin{cases} \upsilon_{\mathtt{V}}(k) \text{ , if } k \in \rho(\mathtt{V})\\ \upsilon_{\mathtt{W}}(k) \text{ , if } k \in \rho(\mathtt{W})\\ 0 \quad \text{ , otherwise}\end{cases}\\
\prec_{\mathtt{X}}(k,l) &\equiv \begin{cases} \prec_{\mathtt{V}}(k,l) \text{ , if } k \in \rho(\mathtt{V}) \wedge l \in \rho(\mathtt{V})\\ \prec_{\mathtt{W}}(k,l) \text{ , if } k \in \rho(\mathtt{W}) \wedge l \in \rho(\mathtt{W})\\ t \quad \text{ , if } k \in \rho(\mathtt{V}) \wedge l \in \rho(\mathtt{W})\\ f \quad \text{ , otherwise}\end{cases}
\end{aligned}$$

The adaptation of the selection on lists σ_l to vague lists is a mapping, that expects a vague list and a three-valued logical predicate as input, and that delivers a vague list as result: $\sigma_{l,3} : \widetilde{\mathbf{List}}(T) \times (T \to \{t,f,u\}) \to \widetilde{\mathbf{List}}(T)$

The semantics is as follows:

$$\begin{aligned}
\sigma_{l,3}(\mathtt{V},P_3) = \mathtt{X}, \text{ with } \quad \Delta_{\mathtt{X}} &= \{(i,n) \in \Delta_{\mathtt{V}} \mid P_3(i) \not\equiv f\}\\
\lambda_{\mathtt{X}}(k) &= \begin{cases} \lambda_{\mathtt{V}}(k) \text{ , if } P_3(i) \equiv t\\ 0 \quad \text{ , otherwise}\end{cases}\\
\upsilon_{\mathtt{X}}(k) &= \begin{cases} \upsilon_{\mathtt{V}}(k) \text{ , if } P_3(i) \not\equiv f\\ 0 \quad \text{ , otherwise}\end{cases}\\
\prec_{\mathtt{X}}(k,l) &\equiv \prec_{\mathtt{V}}(k,l)
\end{aligned}$$

7 Aggregate Functions

Another important type of operations is aggregate functions. Due to space limitations we restrict ourselves to aggregate functions on vague sets here, but the presented techniques can be adapted to vague multisets and vague lists as well.

An aggregate function is a mapping $\alpha : \mathcal{P}(T_1) \rightarrow T_2$, where T_2 is *not* a collection-valued type. This comprises e.g. the maximum, the minimum, the sum, the cardinality, or the arithmetic average of a set.

Taking inaccessibility into account – and thus assuming a *vague* input set instead of a crisp one – we get a vague aggregate function $\alpha_3 : \widetilde{\mathbf{Set}}(T_1) \rightarrow \mathcal{P}(T_2)$. In this case we have to compute a whole set of possible aggregate values each of which could be the correct aggregate value. Formally spoken, $\alpha_3(V)$ is defined as $\alpha_3(V) = \{\alpha(A) \mid A \in V\}$.

If a vague set V is not incomplete, $\alpha_3(V)$ can be determined by the computation of $\alpha(A)$ for each $A \in V$. Because the cardinality of V grows exponentially with the number of uncertain elements in V (namely $|V| = |\mathcal{P}(\widehat{V}_\upsilon \setminus \widehat{V}_\lambda)|$) also the cost for the computation of $\alpha_3(V)$ grows exponentially.

To avoid this effort whenever possible, we identify two classes of aggregate functions for which the vague adaptation can be computed more efficiently. We differentiate set-monotone and element-monotone aggregate functions.

7.1 Set-Monotone Aggregate Functions

There are set-monotone *increasing* and set-monotone *decreasing* aggregate functions. They behave completely analogous.

A set-monotone increasing (decreasing) aggregate function $\alpha : \mathcal{P}(T_1) \rightarrow T_2$ is an aggregate function for which its value does not become lower (higher) when adding elements to the input set: $\forall A, B \in \mathbf{Set}(T_1) : A \subseteq B \Rightarrow \alpha(A) \leq \alpha(B)$.

Examples for set-monotone increasing (decreasing) aggregate functions are the sum of some positive numbers or the maximum (minimum) of a set.

Due to the above definition, the condition $\forall A \in V : \alpha(V_\lambda) \leq \alpha(A) \leq \alpha(V_\upsilon)$ holds for a vague set V and a set-monotone increasing aggregate function α.

By means of this condition, an estimation for the vague aggregate function α_3 on a *complete* vague set V can be computed very efficiently:

$$\alpha_3(V) \subseteq [\alpha(\widehat{V}_\lambda); \alpha(\widehat{V}_\upsilon)]$$

In general this interval also contains values i, for which there is no $A \in V$ with $\alpha(A) = i$. But in most cases this decrease in accuracy will be over-compensated by the increased performance. An example for an application where the decrease in accuracy does not matter at all, might arise if we evaluate a predicate checking whether the maximum of a set is higher than a given value or not.

If the vague set V is *not complete*, we can nevertheless use $\alpha(\widehat{V}_\lambda)$ as a lower (upper) bound for the value of $\alpha_3(V)$ of a set-monotone increasing (decreasing) aggregate function α_3. Unfortunately, the upper (lower) bound for $\alpha_3(V)$ has to be set to the lowest upper bound (greatest lower bound) of the set T_2 in this case.

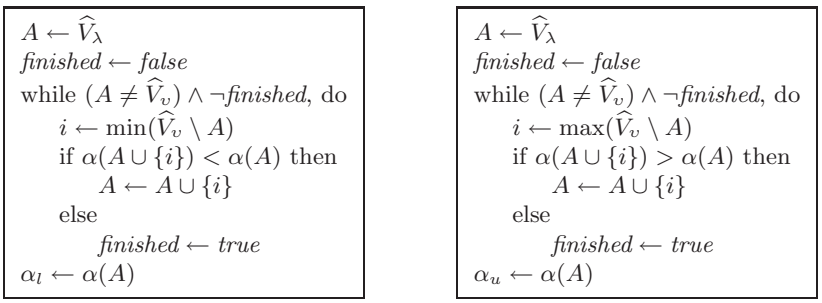


Fig. 2. Algorithms to calculate bounds for the vague version of an aggregate function

7.2 Element-Monotone Aggregate Functions

The second class of aggregate functions for which we propose an efficient algorithm for the computation of (useful) lower and upper bounds is element-monotone aggregate functions. Similarly to set-monotone aggregate functions, there are element-monotone increasing and element-monotone decreasing aggregate functions. Again both subclasses behave analogous.

An element-monotone increasing aggregate function is an aggregate function $\alpha : \mathcal{P}(T_1) \rightarrow T_2$ for which the insertion of a smaller element into the input set yields no higher aggregate value than the insertion of a greater element. Formally: $\forall A \in \mathcal{P}(T_1); i, j \in T_1 \setminus A : i \leq j \Rightarrow \alpha(A \cup \{i\}) \leq \alpha(A \cup \{j\})$

A popular example for an element-monotone increasing aggregate function is the arithmetic average of a set of numbers.

Now we can describe the algorithms for the computation of the vague adaption of an element-monotone increasing aggregate function.

For a complete vague set V a lower bound α_l and an upper bound α_u for the vague adaptation of an element-monotone increasing aggregate function $\alpha_3(V)$ can be computed by the algorithms given in figure 2.

Both algorithms have an asymptotic execution time of $O(n^2)$ in the worst case where n is the number of uncertain elements in V, i.e. $n = |\widehat{V}_v \setminus \widehat{V}_\lambda|$. Fortunately the asymptotic execution time of the algorithms can be reduced to $O(n \log n)$ by previously sorting $\widehat{V}_v$ (at a cost of $O(n \log n)$).

Unfortunately, the described algorithms are only applicable if the vague set V is indeed complete. Otherwise special techniques exploiting specific characteristics of the concrete aggregate function α and the information contained in the descriptive component of the vague input set have to be applied.

References

1. S. Abiteboul, P. Kanellakis, and G. Grahne. On the representation and querying of sets of possible worlds. In *Proc. ACM SIGMOD 1987 Int. Conf. on Management of Data,* pages 34–48, San Franzisco, Cal., USA, 1987. 263

2. L. Amsaleg, M.J. Franklin, A. Thomasic, and T. Urhan. Scrambling Query Plans to Cope With Unexpected Delays. In *Proc. 4th Int. Conf. on Parallel and Distributed Information Systems (PDIS96),* Miami Beach, Florida, Dec 1996. 263
3. K. Basu, R. Deb, and P. K. Pattanaik. Soft sets: An ordinal formulation of vagueness with some applications to the theory of choice. *Fuzzy Sets and Systems,* 45:45–58, 1992. 263
4. P. Bonnet and A. Thomasic. Partial Answers to Unavailable Data Sources, 1997. INRIA Rocquencourt, Rapport de Recherche 3127. 263
5. E. F. Codd. Extending the database relational model to capture more meaning. *ACM Transactions on Database Systems,* 4(4):397–434, 1979. 263
6. European Computer Manufacturers Association, Geneva. *Portable Common Tool Environment - Abstract Specification (Standard ECMA-149),* 1993. 261
7. W. L. Gau and D. J. Buehrer. Vague sets. *IEEE Transactions on Systems, Man and Cybernetics,* 23(2):610–614, 1993. 263
8. G.H. Gessert. Handling missing data by using stored truth values. *ACM SIGMOD Record,* 20(3):30–42, 1991. 263
9. O. Haase and A. Henrich. Error propagation in distributed databases. In *Proc. 4th Int. Conf. on Information and Knowledge Management (CIKM'95),* pages 387–394, 1995. 264, 265
10. O. Haase and A. Henrich. A hybrid representation of vague collections for distributed object management systems. accepted for publication in IEEE Transactions on Knowledge and Data Engineering, 1999. 264, 265
11. A. Heuer and A. Lubinski. Database access in mobile environments. In *Proc. 7th Int. Conf. on Database and Expert Systems Applications, volume 1134 of LNCS,* pages 544–553, Zürich, 1996. Springer. 261
12. T. Imieliński and W. Lipski. Incomplete information in relational databases. *Journal Association for Computing Machinery,* 31(4):761–791, 1984. 263
13. W. Lipski. On relational algebra with marked nulls. In *Proc. 3rd ACM SIGACT–SIGMOD Symposium on Principles of Database Systems,* pages 201–203, Waterloo, Canada, 1984. 263
14. J.M. Morrissey. Imprecise information and uncertainty in information systems. *ACM Transactions On Information Systems,* 8(2):159–180, 1990. 263
15. A. Motro. Accommodating imprecision in database systems: Issues and solutions. *ACM SIGMOD Record,* 19(4):69–74, 1990. 263
16. Z. Pawlak. Rough Sets. *International Journal of Computer and Information Sciences,* 11(5):341–356, 1982. 263
17. A. Thomasic, R. Amouroux, P. Bonnet, O. Kapitskaia, H. Naacke, and L. Raschid. The Distributed Information Search Component (Disco) and the World Wide Web. In *Proc. SIGMOD 1997 Conference,* pages 546–548, Tucson, Arizona, May 1997. 263
18. Y. Vassiliou. Null values in data base managment: A denotational semantics approach. In *Proc. ACM SIGMOD 1979 Int. Conf. on Management of Data,* pages 162–169, Boston, Mass., USA, 1979. 263
19. E. Wong. A statistical approach to incomplete information in database systems. *ACM Transactions on Database Systems,* 7(3):470–488, 1982. 263
20. L. Y. Yuan and D.-A. Chiang. A sound and complete query evaluation algorithm for relational databases with disjunctive information. In *Proc. 8th ACM SIGACT–SIGMOD-SIGART Symposium on Principles of Database Systems,* pages 66–74, Philadelphia, Pa., USA, 1989. 263

Infinite Relations in Paraconsistent Databases *

Nicholas Tran and Rajiv Bagabi

Department of Computer Science, Wichita State University
Wichita, KS 67260-0083, USA
Tel: +1-316-978-3156
Fax: +1-316-978-3984
Email: {tran,bagai}@cs.twsu.edu

Abstract. Paraconsistent information is information that is incomplete and/or inconsistent. A data model for representing and manipulating paraconsistent information in relational databases has recently been developed. Algebraic operators on the underlying paraconsistent relations of this model are generalizations of the usual ones on ordinary relations. However, unlike in the ordinary case, a DBMS based on paraconsistent relations must be capable of handling infinite relations. In this paper, we show this necessity and identify classes of infinite paraconsistent relations whose members can be effectively represented and manipulated. We show that the classes of REGULAR and, under different conditions, CONTEXT-SENSITIVE as well as PSPACE paraconsistent relations are such. We also show that the CONTEXT-FREE and R.E. classes do not have the desired properties, while P, NP, LOGSPACE and NLOGSPACE also probably do not. These results help identify the kinds of relational DBMS that can be constructed for handling incomplete and inconsistent information about tuples.

1 Introduction

One limitation of the relational data model of Codd [9] is its inapplicability to non-classical situations. These are situations involving incomplete, or more importantly, even inconsistent information. Such information often results in databases obtained from combining many databases [5, 18], or in belief systems that are based on beliefs of groups of people.

While incomplete information in databases has been studied extensively (see [16, 10, 14]), inconsistent information has not yet enjoyed similar research attention. Logics dealing with inconsistent information are called *paraconsistent* logics, and were studied in detail by da Costa [11] and Belnap [6]. Their application in computer science has been very limited. The most notable work as yet was by Blair and Subrahmanian [7], who proposed logic programming based on paraconsistent logic. Later, Subrahmanian [17] extended the work to disjunctive deductive databases.

* This research has been partially supported by the National Science Foundation research grant no. IRI 96-28866.

Eder et al. (Eds.): ADBIS'99, LNCS 1691, pp. 275-287, 1999.
© Springer-Verlag Heidelberg Berlin 1999

Recently, Bagai and Sunderraman [2] introduced a data model for representing and manipulating incomplete and inconsistent information in a relational setting. The model is based upon 4-valued *paraconsistent relations* as the fundamental mathematical structures. Algebraic operators are also defined over paraconsistent relations, providing a language for expressing queries in paraconsistent databases. Among other applications of this data model are a method for computing the Fitting model [3] and the well-founded model [4] of deductive databases with incomplete information. Since its introduction, the model has also been extended for temporal paraconsistent information [1] and quantitative paraconsistent information [19].

Paraconsistent relations are strictly more general than ordinary relations, in that for any ordinary relation there is a paraconsistent relation with the same information content, but not *vice versa*. An important property of ordinary relations is that the usual algebraic operators on them preserve finiteness. Due to this property, any relational DBMS that starts with a collection of finite ordinary relations can freely manipulate those relations and is guaranteed to obtain only finite relations as results.

Unfortunately, this property is not shared by paraconsistent relations. Thus, if these structures are employed by a DBMS for incomplete and/or inconsistent information, the system should be able to represent and manipulate *infinite* paraconsistent relations. The chosen class of infinite paraconsistent relations should permit finite representation of its elements, and should be closed under their algebraic operators. Only then could it underlie a paraconsistent DBMS.

In this paper we identify classes of infinite paraconsistent relations that satisfy these two requirements. We call such classes *systems*. The first condition of finite representability restricts our search to subclasses of the recursively enumerable paraconsistent relations. Specifically, we focus our attention to the classes in the Chomsky hierarchy: REGULAR $\subset$ CONTEXT-FREE $\subset$ CONTEXT-SENSITIVE $\subset$ R.E., and the classical space-time hierarchy: LOGSPACE $\subseteq$ NLOGSPACE $\subseteq$ P $\subseteq$ NP $\subseteq$ PSPACE.

We show that among all of these classes, only REGULAR is closed under the algebraic operators, and hence is a system. However, after a simple syntactic transformation on tuples that pads all attribute values with a blank symbol in order to have equal length, we show that CONTEXT-SENSITIVE and PSPACE are systems. On the other hand, we show that none of the smaller classes (LOGSPACE, NLOGSPACE, P, NP) is likely to be a system unless it equals co-NP. These results confirm our intuition in REGULAR as a natural candidate for paraconsistent DBMS and suggest further investigation in subclasses of REGULAR that are *efficiently* closed under the algebraic operations.

The rest of this paper is organized as follows. Section 2 gives a brief introduction to the paraconsistent relations and algebraic operators on them. It also contains a justification for the study carried out in later sections. Section 3 establishes that only REGULAR is a system among all the classes under consideration. Section 4 shows that, under the modified definition of tuples, CONTEXT-SENSITIVE and PSPACE are systems, whereas LOGSPACE, NLOGSPACE,

P, and NP probably are not. Finally, Section 5 summarizes our results, their usefulness and provides directions for further research.

2 Paraconsistent Relations

In this section we give an overview of paraconsistent relations and algebraic operators over them. For a detailed exposition, the reader is referred to [2]. We then provide a motivation for the need to develop new techniques for handling infinite paraconsistent relations.

2.1 Algebraic Operators

Let a *relation scheme* (or just *scheme*) Σ be a finite set of *attribute names*, where for any attribute $A \in \Sigma$, $dom(A)$ is a non-empty *domain* of values for A. We do not distinguish between the semantic values of a domain and syntactic names for those values thereby, as in [16], freely permitting values to appear in syntactic objects, such as queries. A *tuple* on Σ is any total map $t : \Sigma \to \cup_{A \in \Sigma} dom(A)$, such that $t(A) \in dom(A)$, for each $A \in \Sigma$. We let $\tau(\Sigma)$ denote the set of all tuples on Σ.

Definition 1. *A* paraconsistent relation *(p.r.) on scheme Σ is a pair $R = \langle R^+, R^- \rangle$, where R^+ and R^- are any subsets of $\tau(\Sigma)$.*

Intuitively, R^+ may be considered as the set of all tuples for which R is believed to be true, and R^- the set of all tuples for which R is believed to be false. We do not assume R^+ and R^- to be mutually disjoint. A non-empty overlap between R^+ and R^- is essentially contradictory information, and its possibility makes paraconsistent relations model belief systems more naturally that knowledge systems. Also, R^+ and R^- may not together cover all tuples in $\tau(\Sigma)$, giving rise to incompleteness.

As an example, suppose a hospital ward has two patients P_1 and P_2, who are tested for some symptoms s_1, s_2 and s_3. Let results from the first round of tests be captured by the following paraconsistent relation *TEST1* on scheme {P,S}:

TEST1

P_1	s_1
P_1	s_3
P_2	s_1

P_1	s_2
P_2	s_2

The above contains the information that patient P_1 was tested positive for symptoms s_1, s_3 and negative for s_2. Also, patient P_2 was tested positive for symptom s_1 and negative for s_2. Note that the test result of patient P_2 for symptom s_3 is unknown. Such incompleteness of information may be due to various reasons, such as test not conducted, or test results not yet available, or even information ignored because it was not considered important.

Let *TEST2* be the following paraconsistent relation on the same scheme, containing result information from the second round of tests:

TEST2

P_1	s_1
P_1	s_2
P_2	s_2
P_2	s_3

P_1	s_3
P_2	s_3

According to the above paraconsistent relation, patient P_2 was tested both positive and negative for symptom s_3. Such inconsistency of information can be caused by, for example, different tests for symptom s_3 producing different results.

We now define algebraic operators on paraconsistent relations that are generalizations of the usual algebraic operators on ordinary relations, as for example found in [15].

Definition 2. *Let R and S be p.r.'s on scheme Σ. Then,*

(a) *the* union *of R and S, denoted $R \dot{\cup} S$, is a p.r. on scheme Σ, given by*

$$(R \dot{\cup} S)^+ = R^+ \cup S^+, \qquad (R \dot{\cup} S)^- = R^- \cap S^-;$$

(b) *the* difference *of R and S, denoted $R \dot{-} S$, is a p.r. on scheme Σ, given by*

$$(R \dot{-} S)^+ = R^+ \cap S^-, \qquad (R \dot{-} S)^- = R^- \cup S^+.$$

An intuitive appreciation of the union operator may be obtained by interpreting relations as properties of tuples. So, $R \dot{\cup} S$ is the "either-R-or-S" property. Now since R^+ and S^+ are the sets of tuples for which the properties R and S, respectively, are believed to hold, the set of tuples for which the property "either-R-or-S" is believed to hold is clearly $R^+ \cup S^+$. Moreover, since R^- and S^- are the sets of tuples for which properties R and S, respectively, are believed to *not* hold, the set of tuples for which the property "either-R-or-S" is believed to *not* hold is similarly $R^- \cap S^-$. The definition of *difference* and of all other operators defined later can (and should) be understood in the same way.

If Σ and Δ are relation schemes such that $\Sigma \subseteq \Delta$, then for any tuple $t \in \tau(\Sigma)$, we let t^Δ denote the set $\{t' \in \tau(\Delta) : t'(A) = t(A), \text{ for all } A \in \Sigma\}$ of all extensions of t. We extend this notion for any $T \subseteq \tau(\Sigma)$ by defining $T^\Delta = \cup_{t \in T}\, t^\Delta$. We now define some relation-theoretic algebraic operators on paraconsistent relations.

Definition 3. *Let R and S be p.r.'s on schemes Σ and Δ, respectively. Then, the* natural join *(or just* join*) of R and S, denoted $R \dot{\bowtie} S$, is a p.r. on scheme $\Sigma \cup \Delta$, given by*

$$(R \dot{\bowtie} S)^+ = R^+ \bowtie S^+, \qquad (R \dot{\bowtie} S)^- = (R^-)^{\Sigma \cup \Delta} \cup (S^-)^{\Sigma \cup \Delta},$$

where $\bowtie$ is the usual natural join among ordinary relations.

It is instructive to observe that $(R \dot{\bowtie} S)^-$ contains all extensions of tuples in R^- and S^-, because at least one of properties R and S is believed to not hold for these extended tuples. Another interesting observation is that $(R \dot{\bowtie} S)^+ = R^+ \bowtie S^+ = (R^+)^{\Sigma \cup \Delta} \cap (S^+)^{\Sigma \cup \Delta}$, which is the dual of $(R \dot{\bowtie} S)^-$.

Definition 4. *Let R be a p.r. on scheme Σ, and Δ be any scheme. Then, the* projection *of R onto Δ, denoted $\dot{\pi}_\Delta(R)$, is a p.r. on Δ, given by*

$$\dot{\pi}_\Delta(R)^+ = \pi_\Delta((R^+)^{\Sigma\cup\Delta}), \qquad \dot{\pi}_\Delta(R)^- = \{t \in \tau(\Delta) : t^{\Sigma\cup\Delta} \subseteq (R^-)^{\Sigma\cup\Delta}\},$$

where π_Δ is the usual projection over Δ of ordinary relations.

A tuple $t \in \tau(\Delta)$ appears in $\dot{\pi}_\Delta(R)^+$ if any member of $t^{\Sigma\cup\Delta}$ is in $(R^+)^{\Sigma\cup\Delta}$. It should be noted that, contrary to usual practice, the above definition of projection is not just for subschemes. However, if $\Delta \subseteq \Sigma$, then it coincides with the intuitive projection operation. In this case, $\dot{\pi}_\Delta(R)^-$ consists of those tuples in $\tau(\Delta)$, all of whose extensions are in R^-.

Definition 5. *Let R be a p.r. on scheme Σ, $A \in \Sigma$, and $a \in dom(A)$. Then the* selection *of R by $A = a$, denoted $\dot{\sigma}_{A=a}(R)$, is a p.r. on scheme Σ, given by*

$$\dot{\sigma}_{A=a}(R)^+ = \sigma_{A=a}(R^+), \qquad \dot{\sigma}_{A=a}(R)^- = R^- \cup \sigma_{A\neq a}(\tau(\Sigma)),$$

where σ_F is the usual selection of tuples satisfying the boolean formula F from ordinary relations.

For any tuple $t \in \tau(\Sigma)$, attributes A, B, such that $A \in \Sigma$, $B \notin \Sigma$ and $dom(A) = dom(B)$, we let $\rho_{A\to B}(t)$ be a tuple on scheme $\Sigma - \{A\} \cup \{B\}$, given by

$$\rho_{A\to B}(t)(X) = \begin{cases} t(A) \text{ if } X = B, \\ t(X) \text{ otherwise.} \end{cases}$$

We extend this notation for any $T \subseteq \Sigma$ by defining $\rho_{A\to B}(T) = \{\rho_{A\to B}(t) : t \in T\}$.

Definition 6. *Let R be a p.r. on scheme Σ, $A \in \Sigma$, $B \notin \Sigma$ and* $\mathrm{dom}(A) = \mathrm{dom}(B)$. *Then the* renaming *of A by B in R, denoted $\dot{\rho}_{A\to B}(R)$, is a p.r. on scheme $\Sigma - \{A\} \cup \{B\}$, given by*

$$\dot{\rho}_{A\to B}(R)^+ = \rho_{A\to B}(R^+), \qquad \dot{\rho}_{A\to B}(R)^- = \rho_{A\to B}(R^-).$$

As an example of extracting paraconsistent information from a database, consider the query:

> *What symptoms were shown negative by some patient in TEST1 and positive by P_2 in TEST2?*

An algebraic expression for this query is:

$$\dot{-}\,(\dot{-}\,\dot{\pi}_{\{\mathtt{S}\}}(\dot{-}\,TEST1) \,\dot{\cup}\, \dot{-}\,\dot{\pi}_{\{\mathtt{S}\}}(\dot{\sigma}_{\{\mathtt{P}=P_2\}}\,TEST2)).$$

It can be verified that this expression evaluates to the following paraconsistent relation:

s_2

s_1
s_3

While verifying the above, it is especially instructive to observe the role played by the negative part of the definitions of the algebraic operators introduced.

2.2 Infinite Domains

In order to understand the necessity of new techniques for handling infinite domains for paraconsistent relations, we first observe that for any scheme $\Sigma = \{A_1, A_2, \ldots, A_n\}$, $\tau(\Sigma)$ is infinite iff for some i, $dom(A_i)$ is infinite. We next take a brief look at the ordinary relational data model.

An *ordinary* relation on Σ is any subset of $\tau(\Sigma)$. Finite relations are especially interesting as they can be easily stored in databases. Moreover, limiting oneself to finite relations happens to be sufficient for a very large class of real-life database applications. The usual algebraic operators on relations, namely $\cup$, $\cap$, binary $-$, $\times$, different varieties of $\bowtie$, σ, π and $\div$, are well-known. An important and an extremely useful property of these operators is that the class of finite relations is closed under all these algebraic operators. Thus, even if $\tau(\Sigma)$ is infinite, if a DBMS starts with a collection of finite relations, any amount of manipulation of these relations using these algebraic operators will result in only finite relations. It is thus easily possible to stay within just the class of finite relations, even if some of the domains (and thus $\tau(\Sigma)$) are actually infinite.

Unfortunately, paraconsistent relations are not as well-behaved. Let $R = \langle R^+, R^- \rangle$ be called *finite* if both R^+ and R^- are finite subsets of $\tau(\Sigma)$. It can be easily verified that the class of finite paraconsistent relations is closed under the operations $\dot{\cup}$, $\dot{-}$, and $\dot{\rho}$. However, $\dot{\bowtie}$, $\dot{\sigma}$ and $\dot{\pi}$ do not in general preserve the finiteness of their operands. If the domain of some attribute name is infinite, the negative component of the result of any of these operators can be infinite. This is not a surprise because the negative information in most database applications *is* infinite. For example, if John's salary is \$40,000, then the tuple $\langle John, 40000 \rangle$ will be in $\texttt{Employee}^+$, but if the domain `Integer` is associated with the salary attribute, then tuples of the form $\langle John, 39999 \rangle$ and $\langle John, 40001 \rangle$ should be in $\texttt{Employee}^-$.

This becomes an issue in paraconsistent relations because, unlike in ordinary relations, all known (or believed) negative information is stored explicitly. It is therefore not enough to stay within the class of finite p.r.'s. A question thus arises:

Which classes of infinite p.r.'s should we consider?

We need to identify a class of infinite p.r.'s that can effectively underlie a paraconsistent DBMS. Any useful class $\mathcal{C}$ should have the following properties:

1. for any $R \in \mathcal{C}$, both R^+ and R^- should be finitely representable, and
2. $\mathcal{C}$ should be constructively closed under the operations defined in subsection 2.1.

We let a class of p.r.'s with the above properties be called a *system*.

The condition of finite representability implies that we should focus on subclasses of recursively enumerable (r.e.) paraconsistent relations. Two well-known classifications of r.e. languages have been studied extensively in the computational complexity literature [13]. The grammar-based Chomsky hierarchy groups r.e. languages into four subclasses of languages representable by

grammars of increasing expressive power: REGULAR $\subset$ CONTEXT-FREE $\subset$ CONTEXT-SENSITIVE $\subset$ R.E. The Turing-machine-based classification, on the other hand, is a finer division of r.e. languages into subclasses finitely representable by (non)deterministic Turing machines whose space (time) usage is restricted by some well-behaved function. These classes are denoted DSPACE($f(n)$), DTIME($f(n)$), and NTIME($f(n)$), where f is a function that bounds the space (time) usage of the machine. For example, the well-known classes P and NP correspond to DTIME(poly) and NTIME(poly) respectively. Similarly, we have PSPACE = DSPACE(poly), LOGSPACE = DSPACE(log), and NLOGSPACE = NSPACE(log). The following classical space-time hierarchy is well-known: LOGSPACE $\subseteq$ NLOGSPACE $\subseteq$ P $\subseteq$ NP $\subseteq$ PSPACE. Furthermore, REGULAR = DSPACE(constant), and CONTEXT-SENSITIVE = NSPACE(linear).

It is natural to consider classes of paraconsistent relations based on these classifications of r.e. languages. We first define languages based on tuples. For the rest of this paper, we will assume that the domain of any attribute of any scheme is the set Γ^* for some alphabet Γ. Given a scheme $\Sigma = \{A_1, A_2, \ldots, A_n\}$ and a tuple $t \in \tau(\Sigma)$, let $l(t) = \{[A_{i_1}; t(A_{i_1})][A_{i_2}; t(A_{i_2})] \cdots [A_{i_n}; t(A_{i_n})] : \{i_1, i_2, \ldots, i_n\} = \{1, 2, \ldots, n\}\}$, where '[',']', and ';' are new symbols. In other words, a member of $l(t)$ is a string of attribute-value pairs for the function t. Since all permutations are included, $|l(t)| = n!$. Recall from subsection 2.1 that domain values are freely permitted to appear in syntactic objects, such as strings in the above. Now, for any $T \subseteq \tau(\Sigma)$, we let $L(T) = \cup_{t \in T} l(t)$.

Let REGULAR denote the class of all paraconsistent relations R such that both $L(R^+)$ and $L(R^-)$ are regular languages. LOGSPACE, NLOGSPACE, P, NP, PSPACE, CONTEXT-FREE, CONTEXT-SENSITIVE, R.E. are defined analogously. (We also let these names denote classes of languages as usual, but their meanings will always be clear from the context.) Which of these classes of p.r.'s is a system? This is the question we will answer in the next two sections.

3 Closure and Non-closure Results

In this section we show that REGULAR is closed under the algebraic operators and therefore is a system. On the other hand, none of the larger classes is closed under $\dot{\pi}$ or $\dot{\cup}$.

3.1 REGULAR is a System

Theorem 1. *REGULAR is closed under* $\dot{\cup}, \dot{-}, \dot{\bowtie}, \dot{\pi}, \dot{\sigma}_{A=a}$, *and* $\dot{\rho}_{A \to B}$.

Proof. Closure under $\dot{-}$ and $\dot{\cup}$ is immediate from their definitions and the fact that regular languages are closed under ordinary union and intersection.

Closure under $\dot{\sigma}_{A=a}$ and $\dot{\bowtie}$ also follows from closure properties of regular languages. In the case of $\dot{\sigma}_{A=a}$, let R be a regular p.r. on scheme Σ, i.e. the languages $L(R^+)$ and $L(R^-)$ are regular. Define $L_{A=a} = \bigcup_{t \in \tau(\Sigma), t(A)=a} l(t)$,

and $L_{A\neq a} = \bigcup_{t\in\tau(\Sigma),t(A)\neq a} l(t)$. Both $L_{A=a}$ and $L_{A\neq a}$ are regular since the domains are assumed to be regular. Clearly $L(\dot{\sigma}_{A=a}(R)^+) = L(R^+) \cap L_{A=a}$ and $L(\dot{\sigma}_{A=a}(R)^-) = L(R^-) \cup L_{A\neq a}$, and hence $\dot{\sigma}_{A=a}(R)$ is also a regular p.r.

In the case of $\dot{\bowtie}$, let R and S be regular p.r.'s on schemes Σ and Δ respectively. It is straightforward to show that $L((R^+)^{\Sigma\cup\Delta})$, $L((R^-)^{\Sigma\cup\Delta})$, $L((S^+)^{\Sigma\cup\Delta})$, and $L((S^-)^{\Sigma\cup\Delta})$ are all regular languages, and hence $L((R \dot{\bowtie} S)^+)$ $= L((R^+)^{\Sigma\cup\Delta}) \cap L((S^+)^{\Sigma\cup\Delta})$ and $L((R \dot{\bowtie} S)^-) = L((R^-)^{\Sigma\cup\Delta}) \cup L((S^-)^{\Sigma\cup\Delta})$ are both regular languages.

Closure under $\dot{\pi}$ and $\dot{\rho}_{A\to B}$ is proved by modifying the DFA accepting the original relations. The proof is trivial for $\dot{\rho}_{A\to B}$. In the case of $\dot{\pi}$, let R be a regular paraconsistent relation on scheme Σ, and S be a projection of R onto scheme Δ. By definition, $S^+ = \pi_\Delta((R^+)^{\Sigma\cup\Delta})$, and $S^- = \{t \in \tau(\Delta) : t^{\Sigma\cup\Delta} \subseteq (R^-)^{\Sigma\cup\Delta}\}$. For concreteness, let $\Sigma = \{A_1, A_2, \ldots, A_l\} \cup \{B_1, B_2, \ldots, B_m\}$, and $\Delta = \{B_1, B_2, \ldots, B_m\} \cup \{C_1, C_2, \ldots, C_p\}$, and the attributes are all distinct. We will show that $L(S^+)$ and $L(S^-)$ are regular.

Let M^+ be a DFA that accepts $L(R^+)$. We construct a *nondeterministic* finite automaton N^+ to accept $L(S^+)$ as follows: on an input that is a permutation of the attribute-value string $[B_1; s(B_1)]\ldots[B_m; s(B_m)][C_1; s(C_1)]\ldots[C_p; s(C_p)]$ for some $s \in \tau(\Delta)$, N^+ ignores the $[C_1; s(C_1)]$, $\ldots$, $[C_p; s(C_p)]$ pairs and simulates M^+ on a permutation of the attribute-value string $[B_1; s(B_1)]\ldots[B_m; s(B_m)][A_1; t(A_1)]\ldots[A_l; t(A_l)]$ for some $t \in \tau(\Sigma)$ instead. Since the pairs $[A_1; t(A_1)], \ldots, [A_l; t(A_l)]$ can have any values and occur anywhere, N^+ guesses their values and positions *nondeterministically* and simulates M^+ over them using only its finite control. N^+ accepts iff *some* simulation accepts. Clearly N^+ accepts an input iff it is in $L(S^+)$.

Now let M^- be a DFA that accepts $L(R^-)$. We construct an *alternating* finite automaton N^- to accept $L(S^-)$ in the same way we constructed N^+, except that N^- accepts its input iff simulations on *all* possible guessed values for the pairs $[A_1; t(A_1)], \ldots, [A_l; t(A_l)]$ at *all* possible guessed positions lead to an accepting state. Clearly N^- accepts $L(S^-)$. We finally note that nondeterministic and alternating finite automata accept only regular languages [13, 8]. ∎

Since it is known that DSPACE($s(n)$) is the class of regular languages for any functions $s(n) \in o(\log\log n)$ [12], Theorem 1 applies to the class of DSPACE($s(n)$) paraconsistent relations for $s(n) \in o(\log\log n)$ as well.

3.2 LOGSPACE, P, NP, PSPACE, etc. are not Systems

In contrast, we show that for none of the classes of DSPACE($s(n)$) paraconsistent relations where $s(n) \in \Omega(\log\log n)$ is closed under $\dot{\pi}$, and thus none of them is a system.

Definition 7. *Let L be any recursively enumerable language, and M_L be a deterministic one-tape Turing machine that accepts L. At any instant, the* instantaneous description (ID) *of M_L is a binary string of the form xqy, where xy is the current tape content, q is the current state, and the leftmost symbol of*

y is the current symbol visited by the tape head. For each string w in L, the accepting computation of M_L on w *can be encoded in a string* $c^w_{M_L}$ *of the form* $ID_0\#ID_1\#\ldots\#ID_{f_w}$, *where each* ID_i *is the instantaneous description of the machine* M_L *after* i *moves on input* w.

Lemma 1. *The class of DSPACE(s(n)) paraconsistent relations is not closed under* $\dot{\pi}$ *for any recursive function* $s(n) \in \Omega(\log\log n)$.

Proof. Let $K \subset \{0,1\}^*$ be a complete set for R.E., and $\{c^w_{M_K}\}$ be the set of accepting computations of some Turing machine M_K that accepts K. Let $\Sigma = \{A_1, A_2, A_3\}$ and $\Delta = \{A_2\}$ be two schemes, $dom(A_1) = 1^*$, $dom(A_2) = \{0,1\}^*$, and $dom(A_3) = \{0,1,\#\}^*$. Define a paraconsistent relation $R = \langle R^+, \emptyset\rangle$ on Σ, where $R^+ = \{t \in \tau(\Sigma) : t(A_1) = 1^l, t(A_2) = w, t(A_3) = c^w_{M_K}, w \in K,\ |c^w_{M_K}|$ is the smallest divisor of $l\}$. We will show that R is a DSPACE$(\log\log n)$ paraconsistent relation, but the projection S of R onto Δ is not a DSPACE$(s(n))$ paraconsistent relation for any recursive function $s(n)$.

First we show that R is a DSPACE$(\log\log n)$ paraconsistent relation. Construct a Turing machine N such that on an input that is a permutation of the attribute-value string $[A_1; 1^l][A_2; w][A_3; z]$, N computes the smallest i that divides l. This takes only $O(\log(i)) = O(\log\log l)$ space [13]. Next, N verifies that $|z| = i$, which again can be done in $O(\log i)$. Finally, N verifies that $z = c^w_{M_K} = ID_0\#ID_1\#\ldots\#ID_{f_w}$ is the accepting computation of M_K of the given string w. This can done with a two-head DFA, and hence takes at most $O(\log|z|) = O(\log i)$. Clearly, N accepts $L(R^+)$ using space $O(\log i) = O(\log\log|[A_1; 1^l][A_2; w][A_3; c^w_{M_K}]|)$.

Now consider the projection S of R onto scheme $\Delta = \{A_2\}$. We have $L(S^+) = \{[A_2; w] : w \in K\}$, which is nonrecursive, and thus cannot be accepted by any Turing machine bounded in space by a recursive function. ∎

Using the same technique as in Lemma 1, we can show that time-bounded classes of paraconsistent relations are not closed under $\dot{\pi}$.

Lemma 2. *The classes of DTIME(t(n)) and NTIME(t(n)) paraconsistent relations are not closed under* $\dot{\pi}$ *for any recursive function* $t(n) \in \Omega(n)$.

Proof. Define $R = \langle R^+, \emptyset\rangle$, where $R^+ = \{t \in \tau(\Sigma) : t(A_1) = w, t(A_2) = c^w_{M_K},$ and $w \in K\}$, and K and $c^w_{M_K}$ are defined as in Lemma 1. ∎

The following theorem is now easy to verify.

Theorem 2. *The classes LOGSPACE, NLOGSPACE, P, NP, PSPACE and CONTEXT-SENSITIVE are not systems. Also, R.E. is not closed under* $\dot{\pi}$, *and CONTEXT-FREE is not closed under* $\dot{\cup}$.

4 Padded Paraconsistent Relations

The results in the previous section show that only REGULAR is a system. Unfortunately, many normal operations such as comparisons and arithmetic cannot

be performed with regular paraconsistent relations. Such operations would be necessary for a more realistic selection operator, for example, of the form $\dot{\sigma}_{A=B}$ or $\dot{\sigma}_{A\geq B}$.

In this section we propose a simple syntactic transformation on tuples that results in all attribute values to have the same length. This is not a restriction, since there is a one-to-one correspondence between the normal and the new set of tuples.

Let Σ be a scheme. For each tuple $t \in \tau(\Sigma)$, let $\phi(t)$ be the *padded tuple* on Σ obtained by padding the shorter attribute values on the left with a new symbol $\sqcup$ so that each attribute value has length $\max_{A\in\Sigma} |t(A)|$. We let $\beta(\Sigma)$ denote the set of all padded tuples on Σ. Clearly, ϕ is a one-to-one correspondence between $\tau(\Sigma)$ and $\beta(\Sigma)$. A *padded paraconsistent relation on* Σ is a pair $R = \langle R^+, R^-\rangle$ where R^+ and R^- are any subsets of $\beta(\Sigma)$. The class of padded REGULAR (LOGSPACE, NLOGSPACE, etc.) paraconsistent relations are defined in the same way as its normal counterpart.

Under the new definition of tuples, we can show that padded CONTEXT-SENSITIVE and padded PSPACE are closed under the algebraic operators. We also provide strong evidence that the smaller classes of padded paraconsistent relations are unlikely to be systems. Specifically, we show that if the class of padded LOGSPACE (NLOGSPACE, P, NP) paraconsistent relations is closed under the algebraic operators, then the class of LOGSPACE (NLOGSPACE, P, NP) languages equals the class of co-NP languages. A proof of any such equality would be a major breakthrough in complexity theory.

Theorem 3. *Padded CONTEXT-SENSITIVE and padded PSPACE are closed under* $\dot{\cup}$, $\dot{-}$, $\dot{\bowtie}$, $\dot{\pi}$, $\dot{\sigma}_{A=a}$, *and* $\dot{\rho}_{A\to B}$.

Proof. It is easy to see that these classes are closed under $\dot{\cup}$, $\dot{-}$, $\dot{\sigma}_{A=a}$, $\dot{\bowtie}$, and $\dot{\rho}_{A\to B}$. The requirement that each attribute value of a tuple has the same length allows $\dot{\pi}$ to be carried out in linear space. In fact, it can easily be seen that these classes are closed under stronger selection operators such as $\dot{\sigma}_{A=B}$ and $\dot{\sigma}_{A\geq B}$. ∎

As for the smaller classes of padded paraconsistent relations, since logarithmic space is necessary to enforce the requirement that the range of each tuple consists of strings of equal length, we consider only classes padded in space by a function $s(n) \in \Omega(\log n)$. The following theorems show that the smaller classes of padded paraconsistent relations are very unlikely to be systems.

Theorem 4. *Let* $\mathcal{C} \in \{$*LOGSPACE, NLOGSPACE, P, NP*$\}$. *The class of padded* $\mathcal{C}$ *paraconsistent relations is closed under* $\dot{\pi}$ *iff* $\mathcal{C}$ *= co-NP.*

Proof. We prove the theorem for the case $\mathcal{C}$ = LOGSPACE. The other cases are similar. Suppose the class of padded LOGSPACE paraconsistent relations is closed under $\dot{\pi}$. We will show that LOGSPACE = co-NP.

Let $K \subset \{0,1\}^*$ be a complete set for NP, and $\{c^w_{M_K}\}$ be the set of accepting computations of some NP Turing machine M_K that accepts K. Without loss of

generality we may assume that all accepting computations of M_K on input w has length $|w|^m$ for some $m \geq 1$.

Let $\Sigma = \{A_1, A_2\}$ and $\Delta = \{A_2\}$ be two schemes, $dom(A_1) = \{0,1\}^*$, and $dom(A_2) = \{0,1,\#\}^*$. Define a padded paraconsistent relation $R = \langle R^+, \emptyset \rangle$ on Σ, where $R^+ = \{t \in \beta(\Sigma) : t(A_1) = \sqcup^n w, t(A_2) = c^w_{M_K}, w \in K \ \& \ |\sqcup^n w| = |c^w_{M_K}|\}$. First we show that R is a padded LOGSPACE paraconsistent relation. Construct a Turing machine N^+ such that on an input that is a permutation of the attribute-value string $[A_1; \sqcup^n w][A_2; z]$, N verifies that $|\sqcup^n w| = |z|$ and that z is an accepting computation of M_K on w. The whole computation can be done in $\log(|[A_1; \sqcup^n w][A_2; z]|)$, and hence $L(R^+)$ is in LOGSPACE. Now consider the projection S of R onto scheme $\Delta = \{A_2\}$. We have $L(S^+) = \{[A_1; \sqcup^{|w|^k - |w|} w] : w \in K\}$. If $L(S^+)$ is in LOGSPACE, then K is also in LOGSPACE which implies that LOGSPACE = NP = co-NP.

Conversely, suppose LOGSPACE = co-NP = NP. Let R be any padded LOGSPACE paraconsistent relation. For concreteness, let $\Sigma = \{A_1, A_2, \ldots, A_l\} \cup \{B_1, B_2, \ldots, B_m\}$, and $\Delta = \{B_1, B_2, \ldots, B_m\} \cup \{C_1, C_2, \ldots, C_p\}$, and the attributes are all distinct. We will show that $L(S^+)$ and $L(S^-)$ are in LOGSPACE.

Let M^+ be a LOGSPACE Turing machine that accepts $L(R^+)$. We construct an NP Turing machine N^+ to accept $L(S^+)$ as follows: on input that is a permutation of the attribute-value string $[B_1; s(B_1)] \ldots [B_m; s(B_m)][C_1; s(C_1)] \ldots [C_p; s(C_p)]$ for some $s \in \beta(\Delta)$, N^+ ignores the $[C_1; s(C_1)], \ldots, [C_p; s(C_p)]$ pairs and simulates M^+ on a permutation of the attribute-value string $[B_1; s(B_1)] \ldots [B_m; s(B_m)][A_1; t(A_1)] \ldots [A_l; t(A_l)]$ for some $t \in \beta(\Sigma)$. Since the pairs $[A_1; t(A_1)], \ldots, [A_l; t(A_l)]$ can have any values and occur anywhere, N^+ guesses their values and positions *nondeterministically* and simulates M^+ over them, using only a linear amount of space to store the guessed values and positions. N^+ accepts iff *some* simulation accepts. Clearly, N^+ accepts an input iff it is in $L(S^+)$. Thus $L(S^+) \in$ NP = co-NP = LOGSPACE.

Now let M^- be a LOGSPACE Turing machine that accepts $L(R^-)$. We construct a co-NP Turing machine N^- to accept $L(S^-)$ in the same way we constructed N^+, except that N^- accepts its input iff simulations on *all* possible guessed values for the pairs $[A_1; t(A_1)], \ldots, [A_l; t(A_l)]$ at *all* possible guessed positions lead to an accepting state. Clearly N^- accepts $L(S^-)$. Thus $L(S^-) \in$ co-NP = LOGSPACE. ■

5 Conclusions and Future Work

Bagai and Sunderraman [2] proposed a data model for representing and manipulation information in relational databases that is incomplete and/or inconsistent. The model is based upon *paraconsistent* relations with algebraic operators upon them. We showed that in order for a DBMS to employ these relations as its underlying structures, the system needs to be able to handle infinite relations.

The class of infinite relations thus employed should provide for finite representation and effective manipulation of its members. We showed that under a restrictive select operation, $\dot{\sigma}_{A=a}$ for an attribute A and constant a, only the class

of REGULAR paraconsistent relations has the desired properties. However, after a simple syntactic transformation of the tuples, the CONTEXT-SENSITIVE and PSPACE classes exhibit the required properties even for the general selection.

Having identified the classes within which a DBMS may operate, an immediate direction for future work is the complexity analysis of the algebraic operations for the classes that we have shown to be systems. Identifying systems that are *efficiently* closed under the algebraic operations (possibly by using other formal models of computation) will be the first step towards a practical implementation of a paraconsistent DBMS.

References

1. R. Bagai, and M. A. Orgun. A temporal paraconsistent relational algebra for incomplete and inconsistent information. In *Proc. 33rd Annual ACM Southeast Conference*, pages 240–248, 1995.
2. R. Bagai and R. Sunderraman. A paraconsistent relational data model. *International Journal of Computer Mathematics*, 55(1):39–55, 1995.
3. R. Bagai and R. Sunderraman. Bottom-up computation of the Fitting model for general deductive databases. *Journal of Intelligent Information Systems*, 6(1):59–75, 1996.
4. R. Bagai and R. Sunderraman. Computing the well-founded model of deductive databases. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 4(2):157–175, 1996.
5. C. Baral, S. Kraus, and J. Minker. Combining multiple knowledge bases. *IEEE Transactions on Knowledge and Data Engineering*, 3(2):208–220, 1991.
6. N. D. Belnap. A useful four-valued logic. In G. Epstein and J. M. Dunn, editors, *Modern Uses of Many-valued logic*, pages 8–37. Reidel, Dordrecht, 1977.
7. H. A. Blair, and V. S. Subrahmanian. Paraconsistent logic programming. *Theoretical Computer Science*, 68:135–154, 1989.
8. A. K. Chandra, D. K. Kozen, and J. Stockmeyer. Alternation. *JACM*, 28(1):114–133, 1981.
9. E. F. Codd. A relational model for shared data banks. *CACM*, 13(6):377–387, 1970.
10. E. F. Codd. Missing information (applicable and inapplicable) in relational databases. *SIGMOD Record*, 15(4):53–78, 1986.
11. N. C. A. da Costa. On the theory of inconsistent formal systems. *Notre Dame Journal of Formal Logic*, 15:497–510, 1974.
12. J. Hartmanis, P. M. Lewis II, and R. E. Stearns. Hierarchies of memory limited computations. In *Proc. 6th Annual IEEE Symp. on Switching Circuit Theory and Logical Design*, pages 179–190, 1965.
13. J. Hopcroft and J. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, Reading, Mass., 1979.
14. T. Imielinski, and W. Lipski. Incomplete information in relational databases. *JACM*, 31(4):761–791, 1984.
15. P. Kanellakis. Elements of relational database theory. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science, Volume B (Formal Models and Semantics)*, pages 1073–1156. The MIT Press, Cambridge, Mass., 1990.
16. D. Maier. *The Theory of Relational Databases*. Computer Science Press, 1983.

17. V. S. Subrahmanian. Paraconsistent disjunctive deductive databases. *Theoretical Computer Science*, 93:115–141, 1992.
18. V. S. Subrahmanian. Amalgamating knowledge bases. *ACM Transactions on Database Systems*, 19(2):291–331, 1994.
19. R. Sunderraman and R. Bagai. Uncertainty and inconsistency in relational databases. In S. Chaudhuri, A. Deshpande and R. Krishnamurthy, editors, *Advances in Data Management*, pages 206–220. Tata McGraw Hill Publishing, 1995.

Query Rewriting and Search in CROQUE

Joachim Kröger[1], Regina Illner[2], Steffen Rost[2], and Andreas Heuer[1]

[1] University of Rostock, Computer Science Department,
DB Research Group, D–18051 Rostock, Germany
{jo,heuer}@informatik.uni-rostock.de
wwwdb.informatik.uni-rostock.de/Research/CROQUE.engl.html
[2] c/o IBM Deutschland Entwicklung GmbH, Böblingen Programming Lab,
Department 3303/3301, D–71032 Böblingen, Germany
{illner,srost}@de.ibm.com

Abstract. In query optimization, a given query will be transformed by rewrite rules into an equivalent execution plan that is cheaper than the straightforwardly assigned plan according to some cost model. Finding the cheapest of all equivalent execution plans is a challenge since the rewriting of complex queries on the basis of a large set of rewriting rules may potentially span a very large space of equivalent plans. Consequently, one has to either use search strategies to explore (parts of) the search space or some heuristics to prune this space thus making it efficiently traversable.
This paper presents the use of search strategies in the CROQUE project. The adaptation of some common strategies led to the development of a simple but powerful heuristics which is demonstrated by examples executed in the CROQUE prototype. The proposed heuristics can support any random–based search strategy or can be used stand–alone. It may be integrated seamlessly into most of the present query optimizers without almost any effort.

1 Introduction

The CROQUE project[1] [8] is concerned with different aspects of the optimization and evaluation of object–oriented queries. Starting points of all our considerations are queries in ODMG's OQL ([1], formalized in [22]), that are first represented internally in a hybrid approach of calculus and algebra [5] before being transformed using a rule– and cost–based optimizer. The already developed cost model is not further discussed in this paper.

In CROQUE, rule–based rewriting spans a space of equivalent query execution plans. The cost model allows a comparative assessment of all these plans thus defining the search space of CROQUE. By the use of search strategies, this search space is explored. The search is controlled by a heuristics to first explore the most interesting parts of the search space.

[1] CROQUE (*C*ost– and *R*ulebased *O*ptimization of object–oriented *QUE*ries) is a joint research project of the universities of Rostock and Konstanz funded by the German Research Association (DFG) under contracts He 1768/5-2 and Scho 554/1-2.

Eder et al. (Eds.): ADBIS'99, LNCS 1691, pp. 288–303, 1999.
© Springer-Verlag Berlin Heidelberg 1999

In this paper, we present the use of search strategies in the CROQUE project in detail. The adaptation of the common random–based strategies such as random walk, iterative improvement, simulated annealing and two–phase optimization led to the development of a very simple but surprisingly powerful heuristics that we demonstrate using examples executed in the prototypical implementation of CROQUE. The proposed heuristics is independent of the CROQUE project itself and can therefore in principle support any random–based search strategy or be used stand–alone. The heuristics may be integrated seamlessly into most of the present rule–based query optimizers without almost any effort.

Certainly, the discussed heuristics is not for free. Based on the results presented here, we developed another heuristics. Comparing the optimization effort of both approaches obviously leads to a decision against the heuristics proposed here in the case of ad–hoc–query optimization. But considering the relative strengths of both approaches we are thoroughly convinced that merging the ideas behind the two heuristics will result in a yet more powerful outcome. We will have a short look on the mentioned second approach that is presented in more detail in [16] and discussed in [17]. The fusion of both heuristics will only be sketched in this paper since this is an open problem we are just planning to address in our future work.

In the following presentation, we restrict ourselves to the traditional two level approach consisting of a logical and a physical algebra (as introduced in [3]), since the calculus rewriting is done in a more goal–oriented fashion just using a small set of rewriting rules. A long version of this paper is available [14].

The paper proceeds as follows. We begin in Section 2 by discussing the given search space and the adaptations that were necessary for being able to implement the search strategies in our CROQUE prototype. In Section 3, we present our heuristics and some experimental results obtained from examples executed using the CROQUE prototype implementation. We compare our work with related work in Section 4. Finally, we conclude in Section 5 also giving an outlook on further work in the CROQUE project.

2 Search in CROQUE

Query rewriting on logical algebra expressions potentially spans a space of equivalent query expressions and additionally a space of equivalent execution plans, if also a physical rewriting is realized and even physical execution alternatives are considered (as e.g. done in CROQUE with respect to materialized views [4]). This space of equivalent solutions has to be traversed as fast and as cheap as possible by means of search strategies to find a plan as quickly as possible.

In most cases, it is not really necessary to find the best of all equivalent plans but we are interested in avoiding the selection of the worst plans, i.e. we are satisfied if a sufficiently cheap plan is found. Moreover, sometimes it is yet not possible to find the best of all plans in an acceptable amount of search time.

Therefore, we are searching for the plan that has to be used for evaluating the given user query. The process of searching for this plan in CROQUE as presented

here is done after the search space has been generated completely. The decision to do so was made because it is easy to implement the search in this way thus allowing for a rapid prototyping as well as allowing to demonstrate and verify our heuristics easily.

In the following subsections we first have a look on the shape of the search space in CROQUE, then concentrate on presenting the adaptations of the used common search strategies necessary to meet the requirements of this special search space, and finally summarize our experiences with using these search strategies in our prototype system.

2.1 The Search Space

Essentially, we identified three possibilities to select the plan that has to be used for evaluating the given user query:

- *Exhaustive generation of the whole space and then using a search strategy to select a plan.*
- *Heuristic generation of the search space.*
- *Cost–based generation of the search space.*

The last kind of solution was implemented in EXODUS [3] but has shown there to be inefficient and ineffective at least for this special implementation. Therefore, we do not pursue this alternative here. Further research concerning this topic is left for future treatment trying to overcome the inefficiency problem.

The exhaustive generation offers the advantage of supporting rapid prototyping and the evaluation of the implemented search strategies as mentioned. Moreover, a close inspection of the whole search space is possible with negligible effort resulting in a better understanding of how the strategies really work. It provided us with some insights into the weaknesses of the strategies we took into consideration. Beyond this, we were able to compare the strategies on the basis of some different search spaces resulting in an assessment of space shapes.

The obvious drawback of the exhaustive generation is the effort necessary to generate all possible equivalents. Investigating large spaces as needed for the evaluation of some strategies leads to an enormous consumption of ressources. Therefore, the heuristic generation should be prefered if an ad–hoc–query optimization is desired.

Since we are concerned with an offline–evaluation of the search strategies, we nevertheless decided to start with an implementation of the exhaustive generation. The following reflection supported our decision: extending this approach in direction of a heuristic generation is still possible without further work. Thus, we got in the position to be able to combine the advantages of an exhaustive generation with the possible extension of this implementation to the very efficient heuristic generation without loosing results of work done before.

We designed our search space to have two dimensions according to our heuristics presented in Section 3: one dimension of equivalent logical algebra expressions and a second dimension spanned by equivalent physical plans belonging

to each logical expression. In the first prototype of our query optimizer, which provided the basis for the discussion presented here, there have been no physical choices, i.e. each logical expression was transformed in a one–to–one manner into a physical plan. In effect, our search space only consists of one plan dimension. This shape of search space results in the need to adapt the used search strategies.

2.2 Adaptation of Common Search Strategies

Our first prototype has been implemented using the optimizer generator described in [2]. This optimizer generator is based on CRML [23,9], an extension of the functional language SML [19].

In [24], the well–known search algorithms are divided into four categories: deterministic, random–based, genetic, and hybrid algorithms. We started investigating random walk, iterative improvement, simulated annealing, and two–phase optimization as random–based search strategies. These four have been implemented and are also presented in the following subsections. Thereafter, we decided to have a look on the class of search strategies called genetic algorithms. Genetic algorithms are exceptionally well suited to traverse large search spaces of higher dimensions [25] but the realization of this strategy is not possible straightforwardly in contrast to the other strategies used.

Adapting genetic algorithms requires coding the query plans in a suitable fashion. Such a coding function may easily be defined for relational join order optimization but is rather less obvious for complex query expressions as treated in CROQUE. Thus we renounce on further discussing genetic algorithms and concentrate our attention on the adaptation of the random–based strategies to the one–dimensional shape of our search space.

Within all of the algorithms treated in the following, branch–and–bound pruning and hashing of already visited plans (or marking of those plans) can be done. Therefore, we only have to consider plans that are within a given cost limit (mostly: cheaper than plans considered earlier). Moreover, this ensures that no plan is considered twice.

Random Walk (RW). This is the most simple search strategy: in every step, one plan is randomly chosen. The choice does not depend on any former steps. The best plan found during the search process is delivered as result. The search may be restricted by a time limit or the maximum number of plans visited.

RW could easily be implemented in CROQUE by generating a random value and then selecting the plan at the position specified by this value from the list of equivalent plans.

Iterative Improvement (II). Iterative improvement (e.g. described in [13]) starts at a random state in the search space and improves the solution by repeatedly accepting downhill moves (visiting neighboured plans characterized by lower costs) until it reaches a minimum. This is assumed to be a local minimum, if after a given number of trials there has not been found a neighboured plan

with lower costs. It is worth to be noted that this common algorithm may even fail in selecting a local minimum.

The adaptation of II is done in the following way: plans are neighboured to each other iff they are at neighboured positions in the list of equivalent plans. After starting at a random state, it is decided by a random choice to go left or right in the list of equivalent plans. If the algorithm is not able to accept the first plan in the chosen direction due to its higher cost value, the other direction is pursued. In contrast to the generic algorithm our approach does guarantee that a local minimum is found at least.

Simulated Annealing (SA). Like II, simulated annealing [12] goes down the (cost) hills but SA also does accept uphill moves with a certain probability. This probability is a function depending on the number of moves done before, steadily decreasing by time.

SA originates from the idea to reproduce the process of crystallization. Starting from a certain temperature a liquid is gradually cooled down to a minimum value. The main characteristics of this method are the starting temperature and the cooling factor.

We defined the starting temperature to be twice the starting plan cost value. The cooling factor was defined depending on the starting temperature, the actual number of equivalent plans in the space, and the percentage of plans that shall be visited in maximum.

The implementation of SA on a one–dimensional search space is very similar to the one of II. Whenever a move in the chosen direction is not possible in the first step, the other direction is pursued. The algorithm stops if further moves are not possible. The number of moves is used to control the probability of accepting uphill moves. Similar to II, SA is guaranteed to find a local minimum at least.

Two–Phase Optimization (2PO). Two–phase optimization [13] is a combination of the two strategies II and SA. In the first phase, II is used to find a local minimum and in the second phase SA searches the surroundings, being able to do some uphill moves (refer to Figure 1).

2PO may be implemented for one–dimensional search spaces by only prolonging the starting downhill phase of SA, i.e. not considering these moves in the question whether being able to do uphill moves or not. In effect, 2PO would degenerate to a special version of SA. A more favourable solution is to keep track of the right– and leftmost plan visited in the current run. This enables a search in both directions in every step. In principle, this could also be done in SA, thus enlarging the potentially traversed space. The drawback of this kind of solution is the additional overhead demanding for the maintenance of global variables.

2.3 Summary

We search for the plan that has to be used for evaluating the given user query. This search is done after the search space has been generated completely. Due

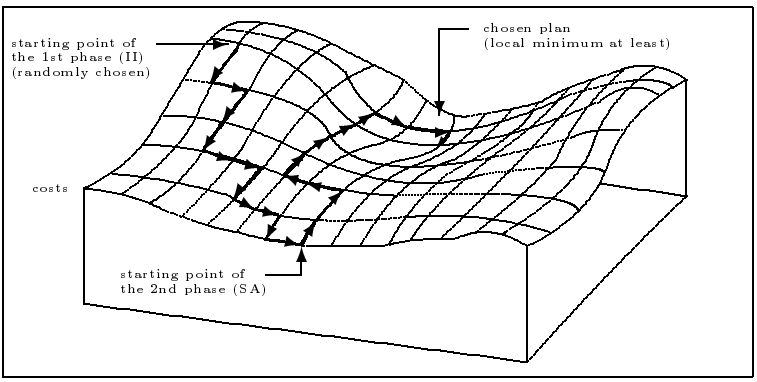


Fig. 1. Two–phase optimization

to the lack of physical choices in our first prototype, our search space has only one plan dimension. This shape of search space results in the need to adapt the used common search strategies.

As could be seen from the above discussion, the adaptation of the strategies is not as complicated as was feared first. This is due to the fact that the definitions of these strategies are given in universally valid generic terms. By determining how the term "neighbourhood" should be defined it is possible to change the generic strategies into algorithms for every kind of search space shape.

Our first prototype comprising the random–based search strategies random walk, iterative improvement, simulated annealing, and two–phase optimization has been implemented using an optimizer generator [2] based on CRML as described in [21,10].

The investigations done using our implementations led to the following observations according to the shape of given search spaces.

- RW is not really satisfying in most cases.
- Search spaces containing some plateaus of the same cost value are not good for II.
- If the cost values mostly alternate, II degenerates to the most simple strategy, RW.
- SA and 2PO are very well suited for large search spaces containing many local minima.
- Search spaces comprising some plateaus of the same cost value and spaces containing cost values alternating most times are bad for SA and 2PO.
- 2PO is more complex but its usage has priority over SA.

These observations motivate our interest in finding heuristics supporting the search as the one presented in the following section.

3 Our Heuristics

All search strategies except random walk heavily depend on the shape of the given search space. A given space can be characterized by two properties. First of all, every search space is obviously characterized by its shape, i.e. the cost value distribution and the number of dimensions. In general, it is not possible to ensure a specific mix of plateaus of the same cost value and the distribution of increasing or decreasing cost values. The number of dimensions may be easily fixed as every value in between one and the maximum number of rewriting rules that are applicable in parallel to a given query expression. Nevertheless, the advantage of a high number of dimensions is questionable as can be seen from the above description of search strategies. At least with regard to random–based search strategies, a one–dimensional search space is more promising since the discovery of local minima can be guaranteed. The shape of a space is none the less hard to influence in a goal–oriented manner.

Another crucial point characterizing a space is the definition of the neighbourhood relation inherently influencing the shape of the space due to its effect on the space generation.

Our heuristics is based on the idea to define a suitable neighbourhood relation whose effect on the space is exploited by a skillful choice of search strategy starting points. In CROQUE, these starting points are the random numbers consumed by the random–based search strategies. All in all, we thus control the search to be as effective as possible.

In the following subsections we first introduce our heuristics. Afterwards, we demonstrate its successful use by some examples, showing how the search may be controlled to become much more effective. We finally summarize our experiences in heuristically controlling the search, thereby also discussing the known drawbacks of our approach.

3.1 The Idea

Often, the neighbourhood relation is directly defined by the application of rewriting rules: two plans are neighboured iff the first plan is the result of applying one rule to the second plan. Whenever two neighboured plans are considered, it is assumed that they are very similar by shape and costs. But this neighbourhood relation is neither complete nor always correct. The same plan may be produced by very different transformation steps. Moreover, the described assumption fails for very effective rewriting rules.

We propose a slightly different way of defining the neighbourhood relation not solving these two problems but being surprisingly powerful as will be shown in Subsection 3.2 by examples.

In general, equivalence rules may be applied in both directions. Since in most cases a heuristically preferred direction of application can easily be distinguished, we only consider directed rules (transformation rules) in the framework of our CROQUE project similar to the VOLCANO approach [6]. The idea of our heuristics is that using more rules for generating a plan will result in a better

plan than using less rules for generation because the rules are only applied in the most favourable direction. More rule applications, i.e. improvements, thus result in better plans.

Therefore, our neighbourhood relation is based on the number of rule applications: two plans are closely neighboured if they are produced by the same number of transformation steps.

Implementing this heuristics is done by ordering the search space during its generation according to the number of realized transformation steps. Every plan therefore contains an integer attribute expressing the number of rules successfully applied during the generation of this special plan. The produced plans are ordered by decreasing factors, i.e. the heuristically better plans are in front (or left) and the original plan is the rightmost one.

Instead of choosing pure, uniformly distributed random values as search strategy starting points, the ordering of plans is exploited by using random values obeying a triangular distribution. By this distribution, the more promising (i.e. ordered more left) plans are privileged.

Our experimental results show some performance gains, i.e. the implementation of our heuristics is more effective than search not supported by our heuristics. Due to space limitations, only two example search spaces will be visited.

3.2 Experimental Results

In this subsection we present some experimental results gained using the implementation of the concepts described above. Due to space restrictions we will neither mention details of our cost model nor present the queries but only use some search spaces for illustration purposes. A long version of this paper is available as [14]. The results presented here have been confirmed by further tests.

The search strategy evaluation was done on the basis of three criteria:

- How often does the search result in finding the global optimum?
  This is an essential criterion expressing the quality of the search. Using random–based search strategies, the result of this criterion depends on the shape of the space, the implemented algorithm, and the generated random values, i.e. especially the quality of the random value generator.
- Of which quality are the other plans selected for execution?
  It is very unlikely traversing only a small part of the search space anyway finding the global minimum every time. Therefore, we are also interested in how often the next best plans are selected. Although the global optimum may have been missed, search results and therefore search strategies, too, may be rated high if a plan nearby the optimum is found most times.
- How many plans have been assessed?
  The number/percentage of plans being visited during search as well as the development of this number if the search space is enlarged may be used to estimate the overall effort of the strategy.

The used search spaces are represented graphically (in Figures 2 and 3.2) by drawing curves connecting the discrete cost values belonging to the listed

plans. Both figures consist of two such curves: in case a) plans produced by the CRML optimizer generator are registered in the search space just in the order they were generated. In case b) the correct ordering according to our heuristics is ensured during list generation. The number of transformation steps done during production of each plan is noticed on the x axis of type b) figures. In all four figures, the best plan is marked by a circle. In type b) figures, an additional line is drawn, showing how the local minima get worse to the right hand side.

Assessing the algorithms is done by carrying out one thousand calls of each algorithm on the unordered and one thousand calls on the ordered search space. The results of these tests are summarized in tables by counting how often plans of which cost value have been selected for execution.

This means, in the tabular representations (in Table 1 and 2 the plans are grouped according to their cost values. The first column ("rank") is used to rank the cost values. Thus, rank value one indicates the global optimal plan(s) found in each space. The column "plans/rank" shows the number of plans per rank, i.e. the number of plans for which the same cost value has been estimated. All the table entries indicate hit rates, i.e. the number of how often plans of this rank have been selected from the considered search space (not ord. = unordered space; ordered = space ordered according to our heuristics) by the considered search strategy (RW, II, and SA).

In the following subsections, three search spaces are used to investigate our heuristics in combination with different search strategies.

Search space containing 44 plans (RW, II, and SA). The search space contains two global optimal plans. These are very close to each other in Figure 2 a) and b). Rewriting the starting plan uses five rules at most. The additional line in Figure 2 b) shows that the local minima at the left hand side are of smaller values than those on the right hand side except for the two optimal plans that have been produced by only three transformation steps.

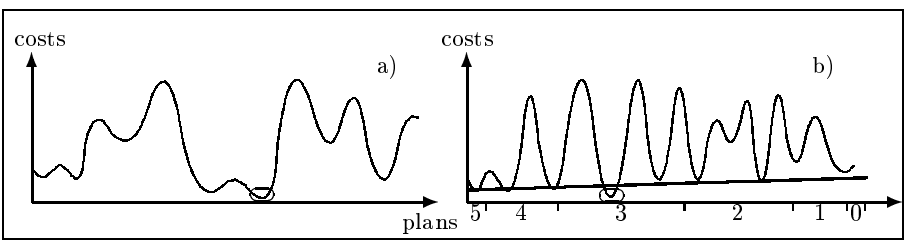


Fig. 2. a) Unordered and b) ordered search space containing 44 plans

Having a look on Figure 2 b) exhibits that there are six more local minima having cost values insignificantly higher than the minimum value.

Table 1 presents the search results for all three search strategies. Random Walk and Iterative Improvement both contain columns labeled "RV" and "ΔRV"

subdividing the search on an ordered space into two cases using uniformly distributed random values and triangular distributed random values, respectively.

Table 1. Search result on the space of Figure 2

rank	plans/ rank	RW			II			SA	
		not ord.	ordered RV	ordered ΔRV	not ord.	ordered RV	ordered ΔRV	not ord.	ordered
1	2	295	279	333	237	339	319	439	248
2	4	405	395	462	308	332	409	356	323
3	2	106	113	205	19	81	272	30	429
4	2	77	79	0	135	58	0	86	0
5	2	34	47	0	10	64	0	25	0
6	2	44	24	0	9	63	0	14	0
7	2	15	24	0	145	5	0	28	0
8	2	13	21	0	0	11	0	0	0
9	2	7	8	0	0	0	0	0	0
10	2	2	6	0	84	21	0	12	0
11	2	2	2	0	0	21	0	1	0
12	2	0	1	0	45	5	0	9	0
13	2	0	1	0	1	0	0	0	0
14	2	0	0	0	7	0	0	0	0
15..21	14	0	0	0	0	0	0	0	0

Comparing the unordered case and the RV column of RW shows the fact that an ordering of plans does not worsen the situation because there is nearly no effect in all. The same holds for II, too. Doing the last step using ΔRV in comparison to RV or the unordered space emphasizes the advantage of our heuristics. All three search strategies clearly benefit from the plan ordering if triangular distributed random values are used since they all result anytime in plans of the best three cost ranks thus avoiding the selection of worse plans.

Note that plans of some cost ranks (e.g. cost rank nine for II in the RV column of Table 1) are never selected neither by II nor by SA. This will happen every time if all the plans of this cost rank are local maxima, since both algorithms ensure the discovery of local minima at least.

Random Walk was adjusted to visit 15% of all plans. II visited 23% and SA 25% (eleven plans) in average. The reason for the higher number of visited plans is that both, II and SA, were programmed by two loops, whereby only the outer one was controlled by a breaking condition related to the overall number of visited plans (15% was the breaking condition here, too). Moreover, the search space investigated by this test is really very small. Using some spaces consisting of more plans showed that II and SA approximated the 15% border value well.

Therefore, we stress that the search result may be positively influenced by exploiting heuristic information to control the search as our approach does according to the number of transformation steps.

Search space consisting of 224 plans (only RW and II). This search space (Figure 3.2) contains two global optimal plans that are very close to each other in both figures, too. There are six plans whose cost values are estimated to be very close to the global optimum. The rewriting takes 6 rules at most. The additional line in Figure 3.2 b) shows, that the local minima at the left hand side are of smaller values than those on the right hand side except for the two optimal plans that have been produced by only four transformation steps.

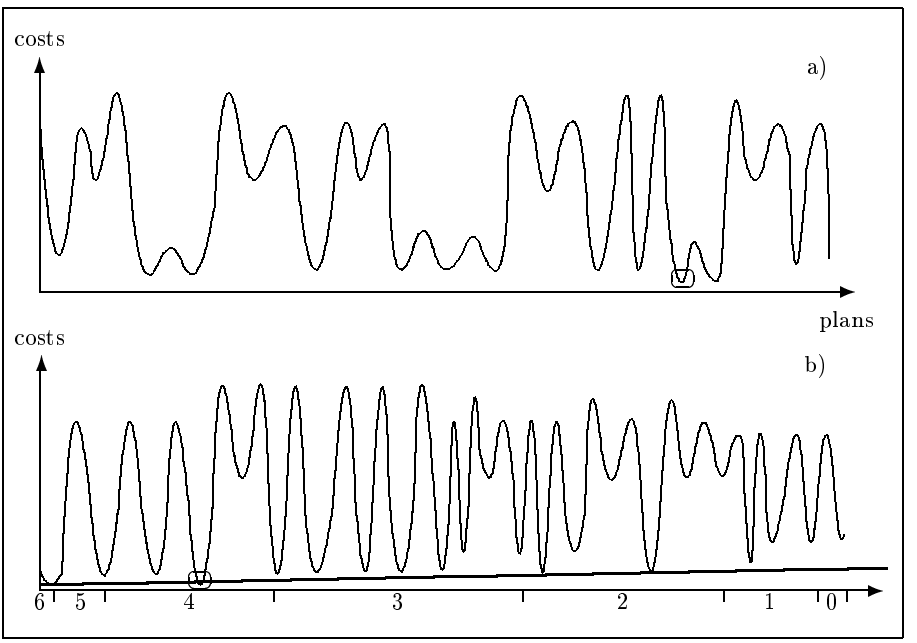


Fig. 3. a) Unordered and b) ordered search space containing 224 plans

Again, the tabular representation (in Table 2) indicates the superiority of our approach. This example is yet more expressive because the search space comprises more plans whereas the plan selection according to the ΔRV column again yields one out of the three best cost values every time.

Moreover, the quality of the result is better every time. A plan of the best rank is found much more often than in the unordered case or by the use of uniformly distributed random values. This is also valid for the second–best and the third–best plan in both cases, RW and II.

Again, Random Walk was adjusted to visit 15% of all plans. Now, II visited 17% of the plans in average. Thus, the inner loop running some time along plans of the same cost value is of minor effect in contrast to the outer loop.

Table 2. Search result on the space of Figure 3.2

rank	plans/ rank	RW			II		
		not ord.	ordered		not ord.	ordered	
			RV	ΔRV		RV	ΔRV
1	2	260	260	408	196	192	261
2	4	388	364	410	163	257	365
3	2	94	108	182	7	111	374
4	2	67	86	0	72	72	0
5	2	51	61	0	9	6	0
6	16	68	54	0	272	151	0
7	8	37	40	0	0	0	0
8	8	35	26	0	179	145	0
9	8	0	1	0	9	0	0
10	2	0	0	0	56	30	0
11	2	0	0	0	5	22	0
12	1	0	0	0	0	3	0
13	8	0	0	0	7	0	0
14	8	0	0	0	5	1	0
15	2	0	0	0	2	6	0
16	2	0	0	0	15	4	0
17..21	1	0	0	0	0	0	0
22	1	0	0	0	2	0	0
23	1	0	0	0	1	0	0
24..45	140	0	0	0	0	0	0

3.3 Summary

As illustrated by the given examples, exploiting our heuristics results in selecting better plans more often than this would happen without the use of this heuristics. But as can also be seen for these "good" search spaces, the pure number of rule applications is not sufficient as a means for plan ordering since the global optimal plan never was the one created using the greatest number of rule applications. This is due to the fact that in this approach all rules are weighted uniformly which does not seem very naturally. How to overcome this inaccuracy is described in [16].

The advantage of our heuristics is the ease of implementation and its power. In fact, the plan ordering corresponds to a depth–first traversal of the potential search space. The obvious main drawback of our approach is the necessarily exhaustive search space generation. For this, a solution avoiding the demand to first span the whole space is sketched in the next section.

4 Related Work

[18] explains how the complexity of optimizing queries with many selection predicates is comparable to join sequence optimization in relational databases. Be-

cause there are many rules describing commutativity and selection pushing, every selection predicate may be arranged at nearly every place in a query. Thus, arranging these predicates is a combinatorial problem like, e.g. join sequence optimization. This justifies the need for far–reaching optimization in the object–oriented context, too.

Until now, we only considered the number of transformation steps that were necessary to generate a plan. This method only seems to be a good choice if the optimization is done offline, i.e. the rewriting is decoupled from query execution. In this case, the time spent for searching the plan that has to be executed is nearly irrelevant. The implementation discussed in Section 3 meets this requirement. There, we focused on the evaluation of search strategies and this task is independent of the time needed to answer the query.

Changing the perspective, a corresponding approach to online (ad hoc) query optimization would consider the quality of the applied rules instead of the quantity of rule applications. The method described in [16] heuristically orders rewriting rules according to their quality (the "optimization potential") and applies them in this order. This is a more promising way for an online optimization where optimization time and query execution time are added yielding the system's overall response time. For this method, the most costly normal form as described in [15] — that may be generated fast by a naive query transformer — seems to be a good starting point. The best rules are applied first gaining the best possible rewriting success very fast, too. Therefore, fast query translation and very fast successful query rewriting result in a low response time.

[20] is concerned with semantic query optimization using rule graphs. Although this topic is not the same as we are concerned with, both have in common that rules are used to come to a more sophisticated solution. [20] states that it is probable that the best reformulated query will be found directly from the rule set rather than by taking longer in query reformulation to search the graph for transitive rules, or build them by successive application of rules to the query. This statement is very similar to the basics of the last–mentioned approach where the rule quality has been used to decide on the rule application and not the application quantity. Therefore, the results of semantic query optimization are partly transferable.

Cascades [7] proposes an offline transitive closure computation to exclude rules from consideration. Therefore, this optimizer generator introduces promise functions to guide the search for the rules that have to be applied. This approach is very similar to ours but in [7] the essential step of defining such functions is left to the database implementor without giving any hint how to do so.

Histograms as presented in [11] for estimation problems seem to be a good basis for an adaptation of the optimizer system by the help of selected statistics.

5 Conclusion and Future Work

We presented the use of search strategies in the CROQUE project. The adaptation of some common strategies according to the characteristics of our search

space led to a very simple but surprisingly powerful heuristics thats benefits have been demonstrated by a detailed discussion of some examples executed in the prototypical implementation of CROQUE.

Whereas the proposed heuristics based on the quantity of rule applications may only be used after the rewriting has completed, the heuristics presented in [16] developed to care more for the quality of the applied rules may be used to control the rewriting. Therefore, a pruning may be achieved during search space generation. This is advantageous but also a much more complex task. To overcome the inherent inaccuracies of both approaches we plan to fuse them. In this way, a more natural consideration of a combination of quality and quantity shall be achieved.

The handling of commutativity rules still is a challenge in both approaches because those rules may have no effect in some algebra only allowing further rules to be applied afterwards.

Acknowledgements

We would like to thank our CROQUE project partners Marc H. Scholl, Torsten Grust, and Dieter Gluche at the University of Konstanz for lots of discussions and hints improving the concepts described here and the quality of the paper, our students Ralf Asmus, Holger Janz, Stefan Paul, Beate Porst, and Denny Priebe for implementing most parts of the first two CROQUE prototypes, and our colleague Jürgen Schlegelmilch giving many hints further improving the paper.

References

1. R.G.G. Cattell, editor. *The Object Database Standard: ODMG-93, Release 1.2.* Morgan-Kaufmann, San Mateo, CA, 1996. 288
2. L. Fegaras, D. Maier, and T. Sheard. Specifying Rule–based Query Optimizers in a Reflective Framework. In *Proc. of the 3rd Int. Conference on Deductive and Object-Oriented Databases*, pages 146–168, New York, December 1993. Springer. 291, 293
3. G. Graefe and D.J. DeWitt. The EXODUS Optimizer Generator. In *Proc. of the ACM SIGMOD Int. Conference on Management of Data*, pages 160–172, San Francisco, CA, USA, May 1987. 289, 290
4. D. Gluche, T. Grust, C. Mainberger, and M.H. Scholl. Incremental Updates for Materialized Views with User-Defined Functions. In *Proc. of the Fifth Int. Conference on Deductive and Object-Oriented Databases (DOOD'97), Montreux, Switzerland, LNCS 1341, Springer*, pages 52–66, December 1997. 289
5. T. Grust, J. Kröger, D. Gluche, A. Heuer, and M.H. Scholl. Query Evaluation in CROQUE — Calculus and Algebra Coincide. In *Proc. of the 15th British National Conference on Databases (BNCOD 15), London, UK, LNCS 1271, Springer*, pages 84–100, July 1997. 288
6. G. Graefe and W.J. McKenna. The Volcano Optimizer Generator: Extensibility and Efficient Search. In *IEEE Conference on Data Engineering 4/1993*, pages 209–218, Vienna, Austria, April 1993. 294

7. G. Graefe. The Cascades Framework for Query Optimization. *Bulletin of the Technical Committee on Data Engineering*, 18(3):19–29, September 1995. 300, 300
8. A. Heuer and J. Kröger. Query Optimization in the CROQUE Project. In *Proc. of the 7th Int. Conference on Database and Expert Systems Applications (DEXA'96), Zurich, Switzerland, LNCS 1134, Springer*, pages 489–499, September 1996. 288
9. J. Hook and T. Sheard. A Semantics of Compile-time Reflection. Technical report, Oregon Graduate Institute, Portland, Oregon, 1993. 291
10. R. Illner. Employment of Simulated Annealing for a Cost-Based Optimization of Object-Oriented Queries. Master's thesis, CS Dept., University of Rostock, June 1996. In german. 293
11. Y. Ioannidis and V. Poosala. Histogram–Based Solutions to Diverse Database Estimation Problems. *Bulletin of the Technical Committee on Data Engineering*, 18(3):10–18, September 1995. 300
12. Y.E. Ioannidis and E. Wong. Query Optimization by Simulated Annealing. In *Proc. of the 13th ACM SIGMOD Int. Conference on Management of Data*, pages 9–22, San Francisco, CA, USA, May 1987. 292
13. Y.C. Kang. *Randomized Algorithms for Query Optimization*. PhD thesis, University of Wisconsin, Madison, October 1991. 291, 292
14. J. Kröger, R. Illner, S. Rost, and A. Heuer. Query Rewriting and Search in CROQUE. Preprint CS–15–98, CS Dept., University of Rostock, December 1998. URL: http://wwwdb.informatik.uni-rostock.de/~jo/CS-15-98.html. 289, 295
15. A. Kemper and G. Moerkotte. Query Optimization in Object Bases: Exploiting Relational Techniques. In J.C. Freytag, D. Maier, and G. Vossen, editors, *Query Processing for Advanced Database Systems*, chapter 3, pages 63–98. Morgan Kaufmann Publishers, 1994. 300
16. J. Kröger, S. Paul, and A. Heuer. On the Ordering of Rewrite Rules. In *Proc. of the Second East-European Symposium on Advances in Databases and Information Systems (ADBIS'98), Poznan, Poland, LNCS 1475, Springer*, September 1998. 289, 299, 300, 301
17. J. Kröger, S. Paul, and A. Heuer. Query Optimization: Ordering Rules? Preprint CS–12–98, CS Dept., University of Rostock, June 1998. URL: http://wwwdb.informatik.uni-rostock.de/~jo/CS-12-98.html. 289
18. F. Ozcan, S. Nural, P. Koksal, M. Altinel, and A. Dogac. A Region Based Query Optimizer Through Cascades Query Optimizer Framework. *Bulletin of the Technical Committee on Data Engineering*, 18(3):30–40, September 1995. 299
19. L.C. Paulson. *ML for the Working Programmer*. Cambridge University Press, 2nd edition, 1996. 291
20. J. Robinson and B. Lowden. Semantic Query Optimisation and Rule Graphs. In *Proc. of the 5th KRDB Workshop*, Seattle, WA, May 1998. 300, 300
21. S. Rost. Analysis of Alternative Search Strategies for a Cost-Based Optimization of Object-Oriented Queries. Master's thesis, CS Dept., University of Rostock, June 1996. In german. 293
22. H. Riedel and M.H. Scholl. A Formalization of ODMG Queries. In *Proc. of the 7th Int. Conference on Database Semantics (DS-7)*, October 1997. 288
23. T. Sheard. Guide to using CRML – Compile-Time Reflective ML. Technical report, Pacific Software Research Center, Oregon Graduate Institute of Science & Technology, Beaverton, Oregon, October 1993. 291
24. Michael Steinbrunn, Guido Moerkotte, and Alfons Kemper. Optimizing Join Orders. Technical Report MIP–9307, Faculty of mathematics and computer sciences, University of Passau, 1993. 291

25. M. Srinivas and L.M. Patnaik. Genetic Search: Analysis Using Fitness Moments. *IEEE Transaction on Knowledge and Data Engineering*, 8(1):120–133, Feb. 1996.

291

Object Query Optimization in the Stack-Based Approach

Jacek Płodzień and Anna Kraken

Institute of Computer Science, Polish Academy of Sciences, Warsaw, Poland
{jpl, kraken}@ ipipan.waw.pl

Abstract. The problem of query optimization in object bases is addressed. A formalized OQL-like query language SBQL is described. SBQL follows the stack-based approach to object query languages. A general method of query optimization based on rewriting is presented. It consists in detecting and factoring out so-called independent subqueries. The approach is based on static analysis of scoping and binding properties for names occurring in a query. The presented method is very powerful and simple to analyze and implement.

1 Introduction

In this paper we address query optimization methods based on rewriting [2, 3, 4, 5]. Rewriting means transforming a query q_1 into a semantically equivalent query q_2 promising better performance. This kind of optimization makes it possible to change the order of operators (e.g. pushing selections before joins), to reduce the complexity of predicates, to remove dead parts of a query (which arise due to querying views), etc. The beauty of rewriting is that optimization algorithms are fast, optimization is made before a query is executed, and the resulting performance improvement is very significant, sometimes several orders of magnitude. Such optimization, however, requires developing criteria to determine whether query q_2 is semantically equivalent to query q_1. This leads to the necessity to define the precise semantics of queries.

To this end many authors have proposed a variety of formal approaches. Among them we can mention F-logic and other approaches extending predicate calculus, object algebras, monoid comprehensions, and others. Generally, however, their technical potential is so far not recognized in detail and there are doubts if they cover vital aspects of object bases and their query languages [8, 10, 11].

At least one important aspect of object database models is neglected by the above mentioned theoretical approaches. It concerns precise semantics of auxiliary names occurring in queries and scoping/binding rules. In contrast, naming, scoping and binding are central issues in our approach to formalization of query languages. Our model is the result of investigations into a uniform conceptual platform for integrated query and programming languages for object bases [8, 9, 11].

In popular programming languages the naming-scoping-binding issues lead to the mechanism and internal data structure known as *environment stack* (ES). It is responsible for the scope control and binding names. A new section of volatile objects (so-called *activation record*) is pushed onto the stack when a procedure/block is

Eder et al. (Eds.): ADBIS´99, LNCS 1691, pp.303 -316, 1999.
© Springer-Verlag Berlin Heidelberg 1999

started, and the section is popped when the procedure/block is terminated. The activation record for a procedure invocation contains volatile variables (objects) that are declared within this procedure, actual procedure's parameters, a return address, and sometimes other data. Binding names follows the "search from the top" rule. The last added section is the first visited during the binding, and objects from some sections remain invisible for the binding (for so-called *lexical scoping*). In implementation ES exists in two forms: as a static stack, which is used at compile-time, and as a dynamic stack, which is manipulated at run-time. These two stack forms are important for this paper.

The stack-based semantics of object query languages is explained in [8]. The idea is that some query operators (called *non-algebraic*) act on the stack in a way similar to invocations of program blocks. For instance, in the query

Student **where** *age* > 20 **and** *year* = 1

the part *age* > 20 **and** *year* = 1 is a block that is (repeatedly) evaluated in environments determined by the tested *Student* objects. For the evaluation of this block, ES is augmented by a new section with references to internal properties (attributes and methods) of an object. After the evaluation this section is popped.

The idea of this paper is based on a simple observation. If a non-algebraic operator (e.g. **where**) is processed, then it creates a new section for bindings on the top of the stack. If all the names in a subquery occurring in the scope of this operator are bound in other sections of the stack, then this subquery is independent of this operator. Such an independent subquery can be factored out before the operator, i.e. executed outside its iteration loop.

The generality and elegance of our idea can be stressed in the following points:

- It covers a very general object model, which includes complex objects, collection-valued attributes, classes, methods, inheritance, encapsulation, roles, and others. We avoid limitations and semantic reefs that plague other approaches [7, 10, 11].
- We cover a quite general case. Our optimization methods are independent of the kind of operator connecting an independent subquery to the outer query.
- An independent subquery that is to be factored out can have arbitrary complexity. It can contain aggregate functions, method calls, joins, quantifiers, etc.

The rest of the paper is organized as follows. In Section 2 we formalize object data structures. In Section 3 we introduce the concept of environment stack. In Section 4 we introduce the formalized OQL-like query language SBQL. In Section 5 we present assumptions concerning static analysis of queries. In Section 6 we describe the method of factoring out independent subqueries. Section 7 concludes.

2 An Abstract Object-Oriented Store Model

The formalization of object-oriented data models is not trivial, because it must be formally simple (to avoid obscure and too sophisticated notions) and at the same time must cover all essential concepts of object bases. The model presented below is some tradeoff. A similar model is assumed in the Stanford Tsimmis project [12].

An *object store* is formed of the structure of objects, OIDs of *root objects* (starting points for querying), and some constraints. Three sets are used to define objects: *I* –

the set of internal identifiers, N – the set of external data names, and V – the set of atomic values (strings, blobs, etc.). Each object has the following features:

- *Internal identifier* (OID); identifiers are not printable;
- *External name* that is used to access an object from a program;
- *Content* of an object which can be a value, a link, or a set of objects.

Let $i, i_1, i_2 \in I$, $n \in N$, $v \in V$. Objects are modeled as the following triples:

- *Atomic objects* as $<i, n, v>$,
- *Link objects* as $<i_1, n, i_2>$,
- *Complex objects* as $<i, n, T>$, where T denotes a set of objects.

The definition is recursive and allows us to create complex objects with an arbitrary number of hierarchy levels. Relationships are modeled through link objects. To model collections we assume that external object names need not be unique. Note that the object concept is relative, in particular, attributes are objects too. We do not deal with the persistence status of objects. Other features of object models (encapsulation, methods, etc.) require extension of the presented model, see [8, 11].

The term "object" denotes exclusively an element of the object store. Our queries never return objects, but some structures built upon references, values and names. We have rejected the *closure property* assumed by other authors, because we consider it semantically invalid and unnecessary. Thus, the dilemma whether queries have to be "object preserving" or "object generating" becomes irrelevant.

Example database

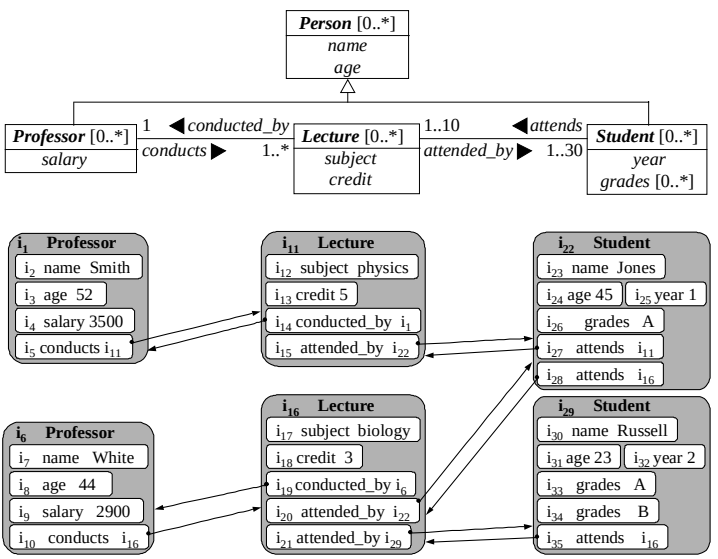


Fig. 1. Database schema and tiny database

Fig.1 presents a schema in a bit extended UML, and a tiny database built upon the schema. We assume that class names are also names of corresponding objects. Class/attribute name is followed by the cardinality of a given object/attribute, unless it is 1. Objects *Professor*, *Lecture* and *Student* are root objects.

3 Stacks and Bindings

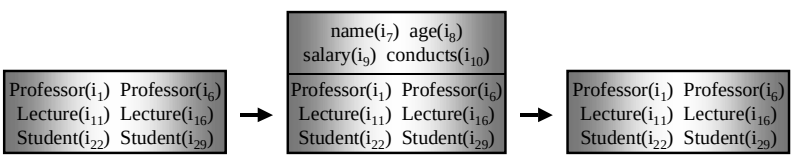
In programming languages the environment stack accomplishes the abstraction principle, which allows the programmer to consider the currently written piece of code to be independent of the context of its possible use. The stack makes it possible to associate parameters and local variables with a particular procedure (function, method) invocation. Thus safe nested calls are possible, including recursive calls. The stack is also necessary for strong typing, encapsulation, inheritance and overriding.

In our approach the stack has a new function: processing queries acting on the object store defined in the previous section. Thus some changes of its construction are necessary. In contrast to programming languages we assume that the object store and the stack are separate data structures. The main reason for this assumption is the fact that the same object can be referred from different stack sections.

The stack consists of *sections* that are sets of *binders*. Binder is a concept that allows us to explain and describe various naming issues that occur in object models and their query languages. Formally, a binder is a pair $n(x)$, where n is an external name, and x is a reference, a value, or some complex structure (a table; defined later).

The binding follows the "search from the top" rule: during the binding of the name n we are looking for a binder $n(x)$ that is closest to the stack top. The result of the binding is x. To cover bulk data structures of our store model we assume that binding is multi-valued: if the relevant section contains more binders whose names are n: $n(x_1)$, $n(x_2)$, $n(x_3)$,..., then all of them form the result of the binding. In such a case binding n returns a collection $\{x_1, x_2, x_3,...\}$.

Some modification to the binding rules is necessary to take into account inheritance. If n_1 is the name of an object from class n_1, the class n_1 has a sub-class n_2, and ES contains a binder $n_2(x)$, then the binding of n_1 is successful and returns x. For instance, if we bind name *Person*, and ES contains the binder *Professor*(i_6), then the binding returns i_6. This rule is generalized for multi-valued bindings, too.



The initial state of ES

The state of ES during evaluation of *p* in query: *Professor* **where** *p* in the second iteration loop

The final state of ES

Fig. 2. States of ES

In Fig.2 we present stack states for the tiny database from Fig.1. During evaluation of queries the stack is growing and shrinking according to query nesting. Assuming no side effects in queries (no calls of updating methods) the final ES state is exactly the same as the initial state.

Object-oriented data structures and complex queries/programs acting on them may create complex states of ES. In Fig.3 we present a state of the stack concerning the binding name *X* from the **where** clause of some query. *X* can be the name of an *Professor* object attribute, the name of a method from the *Professor* class, the name of

a method from the *Person* class, the name of a root database object, the name of a view, etc. The system is trying to bind *X* to the proper entity of the environment, following the order presented in Fig.3 by arrows. For simplicity in this paper we abstract from stack sections storing properties of classes. For the same reason (with no loss of generality) we will treat all "bottom" ES sections (immutable during execution of a particular query) as one section.

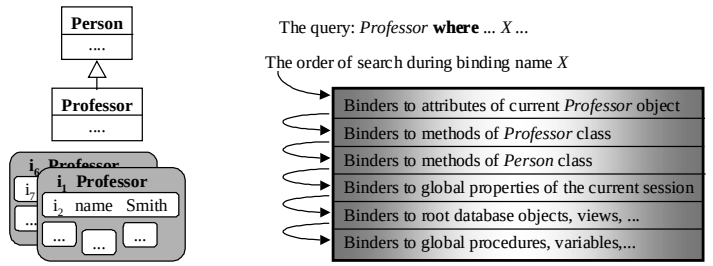


Fig. 3. More complex state of ES

4 Formalized Query Language SBQL

SBQL (*Stack-Based Query Language*) is a formalized language implemented in Loqis [9]. All basic constructs of OQL [6] have counterparts in SBQL. SBQL avoids the syntactic sugar and orthogonal query operators are separated as far as possible.

4.1 SBQL Syntax

- A single name or a single literal is a query.
- If q is a query, and σ is a unary operator (e.g. sum, count, distinct, sin, sqrt, ...), then $\sigma(q)$ is a query. The operator defining a name is a unary operator too (parameterized by the name); we use for it the typical syntax q **as** n (where $n \in N$).
- If q_1 and q_2 are queries, and θ is a binary operator (for instance: **where**, (dot), |, +, =, <, **and**, **union**, ×, ...), then $q_1\ \theta\ q_2$ is a query. Quantifiers are binary operators too, but we apply to them the traditional prefix syntax: $\forall q_1(q_2)$ and $\exists q_1(q_2)$.

With the exception of typing constraints (which are implicit in this paper) we assume orthogonality of operators. For instance, *Professor*, *name*, *age*, "White" are atomic queries. We can build from them complex queries such as:

(*Professor* **where** (*name* = "White")).*age*

In contrast to SQL and OQL, SBQL queries have a very unusual property: they can be decomposed into subqueries, down to atomic ones, connected by unary or binary operators. This is due to the fact that all queries in SBQL are evaluated relatively to the state of ES.

4.2 Results of SBQL Queries

We assume that each SBQL query (or subquery) returns a *table*, understood as a *bag of rows*. Such a table can be empty, can contain a single row, or can contain an arbitrary number of rows. All rows in a table have the same type. A row can contain atomic values $v \in V$, identifiers $i \in I$ (in this context we call them *references*), and binders $n(x)$, where $n \in N$ and x is a table. Note that the definition is recursive.

If a table has one row and one column, it is called a 1×1 table. For simplicity we make no difference between such a table and the element stored in it. In Fig.4 we present tables which can be returned as results of queries, cf. Fig.1. They represent the results of queries 2+2, *Student*, *Professor* **as** *p*, etc.

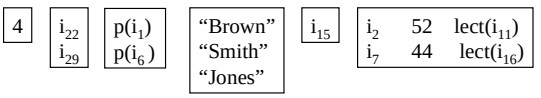


Fig. 4. Sample result table

4.3 SBQL Semantics

To define the operational semantics of queries we introduce an auxiliary stack QRES (*Query RESults*) for storing all temporary and final results of queries. Its elements are tables, as defined in the previous subsection. At the beginning the stack is empty.

The recursive procedure *eval* is used to define the operational semantics of SBQL [8]. *eval* maps a syntactically correct query and a machine state to a result table pushed onto the top of QRES. The machine state consists of the state of the object store and the state of ES. *eval* is defined by cases:

- For query *l*, where *l* is a literal denoting an atomic value $\underline{l} \in V$, *eval*(*l*) pushes the 1×1 table {<$\underline{l}$>} onto QRES. E.g. query 2 pushes the 1×1 table containing 2.
- For query *n*, where $n \in N$, *eval*(*n*) binds name *n* and pushes the result of the binding onto QRES as a single-column table. For instance (cf. Fig.1), query *Student* pushes onto QRES the second table shown in Fig.4.
- Queries can be connected by *algebraic* and *non-algebraic* operators. The difference between algebraic operators and non-algebraic operators is whether they modify ES during evaluation or not.

An operator is algebraic if ES is not modified. Algebraic operators include numerical and string operators, Boolean **and**, **or**, **not**, aggregate functions, set, bag and sequence operators, Cartesian product (denoted by comma), etc. Let $q_1 \; \Delta \; q_2$ be a query consisting of two subqueries connected by a binary algebraic operator Δ. Procedure *eval* takes q_1 and pushes its result table onto QRES, then does the same with q_2, performs Δ on the two top tables of QRES, and finally pops QRES two times and pushes the final result. (Similarly for a unary operator.)

Some algebraic operators implicitly call the dereferencing operator. For example, in query *age* > 40 the first subquery *age* returns a 1×1 table with a reference, say i_8. The operator > calls dereferencing, which changes this reference into the value 44.

The definition of an auxiliary name *n* (having the syntax *q* **as** *n*) is an algebraic operator, too. The operator makes binders by assigning name *n* to each row returned by *q*. For instance, if query *Professor* returns the table {<i_1>, <i_6>}, then the query *Professor* **as** *p* returns the table {<$p(i_1)$>, <$p(i_6)$>}.

If θ is non-algebraic, then $q_1 \theta q_2$ is evaluated as follows. For each row *r* of the table returned by q_1 the subquery q_2 is evaluated. Before the evaluation ES is augmented by a new section determined by *r*. After the evaluation ES is popped to the previous state. A partial result is the combination of *r* and the table returned by q_2. The final result is the combination of the partial results; the kind of combination depends on θ. Note that non-algebraic operators do not follow the basic property of algebraic expressions, which allows independent evaluation of operators' arguments.

A new stack section is determined by row *r* in the following way. Assume *r* consists of a single object reference, e.g. i_6 (cf. Fig.1). The new stack section contains binders to all the internal properties of the object: for i_6 the new section will contain binders *name*(i_7), *age*(i_8), *salary*(i_9), *conducts*(i_{10}); cf. Fig.2. Similarly, if *r* consists of a single reference to a link object, then the new stack section contains the binder of the object the link points to (e.g., *Lecture*(i_{16}) for i_{10}).

The idea is generalized. If *r* contains an object reference, then ES is additionally augmented by sections containing binders to internal properties of its classes, as shown in Fig.3. If *r* contains binders, they are copied (without changes) to the ES section built for *r*. Other kinds of elements of *r* are neglected. The section is the union of all binders that are induced by elements of *r*. This semantics is a uniform basis for the definition of several non-algebraic operators of OQL-like languages:

- **Selection**: q_1 **where** q_2, where q_1 is any query, and q_2 is a Boolean-valued query. If q_2 returns *TRUE* for a particular row *r* returned by q_1, then *r* is an element of the final result table; otherwise it is skipped.
- **Projection, navigation**, **path expression**: $q_1.q_2$. The result table is the union of tables returned by q_2 for each row *r* returned by q_1. This construct covers generalized path expressions [1]; e.g. $q_1.q_2.q_3.q_4$ is understood as $((q_1.q_2).q_3).q_4$.
- **Dependent join**: $q_1 \mid q_2$. A partial result for a particular *r* returned by q_1 is a table obtained by a concatenation of the row *r* with each row returned by q_2 for this *r*. The final result is the union of the partial results.
- **Quantifiers**: $\forall q_1(q_2)$ and $\exists q_1(q_2)$, where q_1 is any query, and q_2 is a Boolean-valued query. For $\forall$ the final result is *FALSE*, if q_2 returns *FALSE* for at least one row *r* returned by q_1; otherwise the final result is *TRUE*. Similarly for $\exists$.

SBQL can be considered a formalized version of OQL. Below we show examples of equivalent OQL and SBQL queries. We assume that each interface *C* in ODL [6] has defined an extent *Cs*, e.g. an interface *Student* defines an extent *Students*.

Get Russell's grades:
OQL: **select** *s.grades* **from** *Students* **as** *s* **where** *s.name* = "Russell"
SBQL: ((*Student* **as** *s*) **where** *s.name* = "Russell").(*s.grades*)

Get names and salaries of professors over 50:
OQL: **select struct**(*n*: *p.name*, *s*: *p.salary*) **from** *Professors* **as** *p* **where** *p.age* > 50
SBQL: ((*Professor* **as** *p*) **where** *p.age* > 50).(*p.name* **as** *n*, *p.salary* **as** *s*)

For each lecture get subject, professor name, and the number of attenders:
OQL: **select** *l.subject*, *p.name*, count(**select** *x* **from** *l.attended_by* **as** *x*)
from *Lectures* **as** *l*, *l.conducted_by* **as** *p*
SBQL: ((*Lecture* **as** *l*) | (((*l.conducted_by*).*Professor*) **as** *p*)).
(((*l.subject*), (*p.name*)), count((*l.attended_by*).*Student*) **as** *x*).*x*))

In Fig.5 we present stages of the evaluation of a simple query.

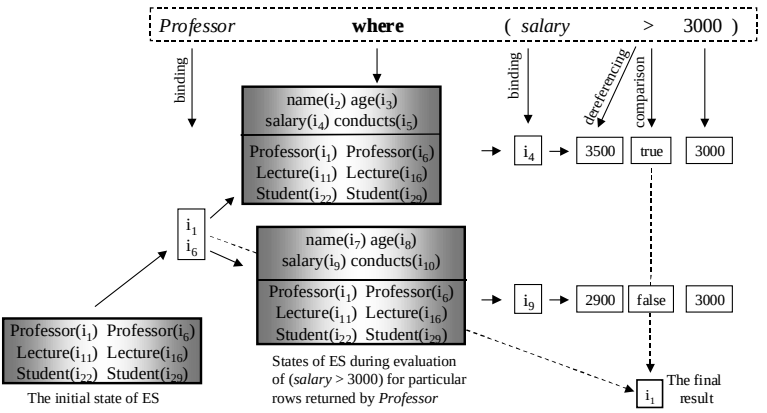


Fig. 5. Evaluation of query *Professor* **where** *salary* > 3000

5 Static Analysis of Queries

Because the optimization we are dealing with is static, we simulate at compile-time the behaviour of the run-time query mechanism. The following data structures are managed during static analysis:

- **Database schema graph** generated from the database schema.
- **Static environment stack**. It models statically the behaviour of the run-time ES, in particular, binding names and opening new scopes by non-algebraic operators.
- **Static result stack**. It models statically the behaviour of the run-time QRES.

Two last structures simulate query evaluation as precisely as possible during the static analysis. This phase checks types, generates a syntactic tree, and associates a particular name occurring in the query with an ES section where the name will be bound. This information is essential for our rewriting method.

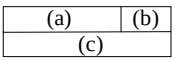
5.1 Database schema graph

The nodes of a schema graph contain the definitions of database entities. The edges model relationships between them. There are three kinds of such relationships:

- *is_owner_of* - between the definition of a complex object and the definition of its subobject. It is represented as a solid arrow.

- *points_to* - between the definition of a link object and the definition of the object it points to. It is represented as a dashed arrow.
- *inherits_from* - between the definition of an object and the definition of another object. It is represented as a double-line arrow.

A graph node is represented as follows:



- (a) – external name of an entity, e.g.: *Student*, *Professor*, *salary*;
- (b) – cardinality of an entity. It says how many instances this entity can potentially have within its parent: 1 - exactly one; 0..1 - optional; 0..* - at least zero, etc.
- (c) – type of the value of an object: an atomic type, a link type (dashed arrow) and a complex type (several solid arrows).

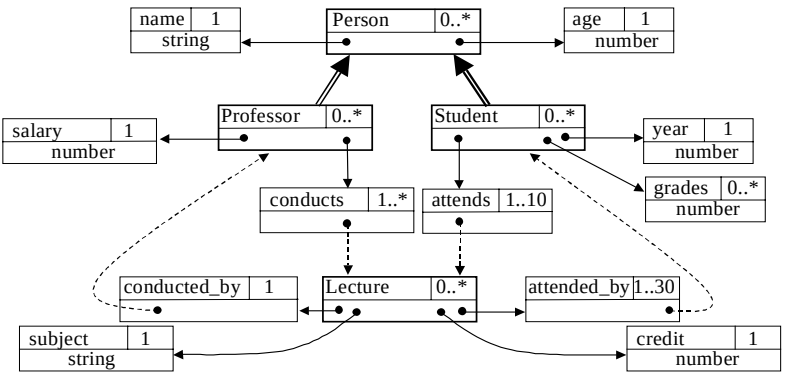


Fig. 6. Schema graph

Rectangles with thick frames denote root objects. In implementation the schema graph can also contain other data. Fig.6 presents the schema graph of our database. A graph node is identified by an internal identifier. As a convention, in this paper such identifiers are denoted ref_n, where n is the external name associated with a node.

5.2 Static stacks

States of the static stacks approximate states of the run-time stacks. Tables on QRES and stack sections on ES are modeled as *table signatures* and *section signatures*, respectively (or *signatures*, for short). A section signature consists of *static binders*, which have the following format:

- *n*(*type*) – if at run-time the corresponding binder is the named value of an atomic type. Then *n* is the name of this binder, and *type* is the type of the value;
- *n*(*ref*) – if at run-time the corresponding binder is the named identifier of an object. Then *n* is the binder name and *ref* is the identifier of a graph node;
- *n*(*signature*) - for more complex static binders (not considered in this paper).

A table signature on the static QRES stack is more complex. It is a sequence of static binders, unnamed references to nodes in the schema graph and unnamed types of atomic values. All elements of static stacks can be associated with cardinalities.

5.3 Static Analysis Algorithm

The general algorithm behaves in the same way as the algorithm of run-time evaluation, with the following differences:

- Operations are performed on signatures rather than on run-time entities.
- The signature of query *l*, where *l* is a literal, is the inferred type of *l*, e.g. *integer*, *string*, *boolean*, etc.
- The table signature of the query *n*, where *n* is a name, is inferred from static ES. Namely, the system looks for a static binder *n*(*x*). The resulting signature is <*x*>.
- If *q* has the signature <*q_sig*>, then query *q* **as** *n* has the signature <*n*(*q_sig*)>.
- The signature of query $q_1 \Delta q_2$, where Δ is algebraic, is a composition of signatures of its components, according to the type inference rules for the operator Δ.
- The signature of a new section (sections) pushed onto the static ES stack is built based on the elements of the signature of the static QRES table that is processed by a non-algebraic operator in a way similar to that at run-time. For example, if the signature of the table is $<ref_{Student}>$, then the signature of the new ES section is $<name(ref_{name}), age(ref_{age}), year(ref_{year}), grades(ref_{grades}), attends(ref_{attends})>$.
- The signature of the query $q_1 \ \theta \ q_2$, where θ is non-algebraic, is composed of the signatures of q_1 and q_2 in the way determined by the operator θ. For instance, if θ is the join, then the resulting signature is the union of signatures of its arguments.

An example of static query analysis is presented in Appendix.

6 Factoring Out Independent Subqueries

According to the SBQL semantics, at run-time each of the non-algebraic operators in a query opens its section on ES, and each name occurring in a query is bound in some section of that stack. During static evaluation each non-algebraic operator is assigned a number denoting the number of the section it opens, and each name in a query is assigned two numbers: the size of the stack when this name is bound and the number of the section in which this name is bound.

The independent subquery method consists in analyzing the numbers of those ES sections in which particular names occurring in a query are bound. It turns out that if all the names in a subquery are not bound in the section opened by the currently evaluated non-algebraic operator, then it can be evaluated earlier than it results from the textual place of this subquery in the query.

Let's consider the simple query "Get lectures whose credits are higher than the credit of physics":

Lecture **where** *credit* > ((*Lecture* **where** *subject* = "physics").*credit*) (**1**)

Let's notice that the subquery returning the credit of physics:

(*Lecture* **where** *subject* = "physics").*credit* (**1a**)

is evaluated for each object *Lecture* existing in the database, while it is enough to calculate it just once. We say that this subquery is independent of its direct non-algebraic operator (external **where**).

Let's examine the stack sizes and binding levels for the names, and the numbers of sections that are opened by the non-algebraic operators from this query (recall that bottom sections are treated as one section):

Lecture **where** *credit* > ((*Lecture* **where** *subject* = "physics").*credit*)
(1,1) 2 (2,2) (2,1) 3 (3,3) 3 (3,3)

As we can see, none of the names in the subquery (1a) is bound in section 2 opened by external **where**. Therefore the subquery (1a) is independent of that operator and can be calculated before it opens its section (before the evaluation of the entire query).

To express it in the textual form of the query, the independent subquery is *factored out*. In SBQL this means the following. First, a new unique auxiliary name is chosen. It will be used as the name of the result of the independent subquery. Then, the subquery is put before the left subquery of the non-algebraic operator it is independent of, and connected to the rest of the query by the dot operator. Finally, the auxiliary name is put in the previous place of this subquery. After factoring the subquery (1a) out, the query (1) will have the following form:

(((*Lecture* **where** *subject* = "physics").*credit*) **as** *c*).(*Lecture* **where** *credit* > *c*)

Now the subquery (1a) is evaluated first and its result is named *c*. Then this name is put in the query (1) in the previous place of (1a). In the above query the second dot operator pushes the binder named *c* onto ES.

Consider the query "Get lectures along with the number of students whose names are the same as professor's name":

Lecture | count((*attended_by*.*Student*) **where** **(2)**
(1,1) 2 (2,2) 3 (3,3) 3
name = (*conducted_by*.*Professor*.*name*))
(3,3) (3,2) 4 (4,4) 4 (4,4)

and its subquery *conducted_by*.*Professor*.*name*. It is independent of the operator **where,** since all the names in this subquery are bound in sections 2 and 4, while **where** opens section 3. However, it is dependent on the operator |, which opens section 2 (the name *conducted_by* is bound in that section). So the subquery can be evaluated before the operator **where**, but not before the operator |. Therefore the query (2) can be rewritten as follows:

Lecture | count(((*conducted_by*.*Professor*.*name*) **as** *n*).
((*attended_by*.*Student*) **where** *name* = *n*))

The general rule is as follows. Consider query $q_1\ \theta\ q_2$, where θ is any non-algebraic operator (**where**, dot, |, quantifier, etc.). Let q_2 have the form $\alpha_1\ q_3\ \alpha_2$, where q_3 is any syntactically correct subquery connected by arbitrary operators to the rest of q_2 (parts α_1 and α_2). If q_3 is independent of the operator θ (i.e., none of its names is bound in the ES section opened by θ) and returns a table with exactly one row, then the query can be rewritten to the form: $(q_3$ **as** $x).(q_1\ \theta\ (\alpha_1\ x\ \alpha_2))$

The method works for the case, when q_3 returns an arbitrary table. To this end we introduce another naming operator **group as**, which names the entire query result.

The semantics of this operator is simple: if q returns any table t, then q **group as** n returns the single binder $n(t)$. Our scoping-binding rules remain unchanged.

The method explained above can be generalized. A given subquery can be independent not only of its direct non-algebraic operator, but also of several non-algebraic operators. Let's consider the query

(*Lecture* **as** *l*) |
(1,1) 2
(*l* .(*attended_by*.(*Student* **where** *age* > (*l* .*conducted_by*.*Professor*. *age*))))
(2,2) 3 (3,3) 4 (4,4) 5 (5,5) (5,2) 6 (6,6) 6 (6,6) 6 (6,6)

which for each lecture gets those students who are older than professor. When the evaluation of the selection predicate starts, there are five sections on ES. Analyzing the binding levels we can see that the part *l.conducted_by.Professor.age* is independent not only of the nearest operator which modifies the section at the top of ES (i.e. of the operator **where**), but also of the other non-algebraic operators (that opened sections 4 and 3) and is dependent on the join operator. Thus the subquery can be factored out of several non-algebraic operators:

(*Lecture* **as** *l*) | (((*l.conducted_by.Professor.age*) **as** *a*).
(*l*.(*attended_by*.(*Student* **where** *age* > *a*))))

The method can be applied to arbitrary non-algebraic operators. We can factor out independent subqueries from complex queries, where the subqueries occur in the scope of quantifiers, joins, dots, etc. They can also occur as parameters of aggregate functions, methods, etc. The method is also independent of the complexity of the subqueries: they can contain view invocations, method calls, arithmetic and aggregate functions, etc. As far as we know such general rewritings were not considered in other formal approaches, e.g. in object algebras.

The Distributivity Property

Much stronger rewriting rules we can obtain if we assume that some non-algebraic operators are distributive. The property is defined as follows.

Definition. A non-algebraic operator θ is *distributive*, if for any query $q_1 \ \theta \ q_2$ the result can be calculated as the union of all results of $r \ \theta \ q_2$, where r is a row returned by q_1 (we assume that θ can be naturally applied to a row, as defined previously); the final result is always the union of partial results taken for all rows returned by q_1.

Proposition 1. Operators **where**, dot and dependent join are distributive.

The proof follows from the definition of these operators, which is based on the union of partial results obtained for every row returned by the query occurring on the left side of such an operator. Quantifiers are not distributive (as well as other non-algebraic operators, not considered here).

Proposition 2.

$(q_1 \mid q_2) \mid q_3$ is equivalent to $q_1 \mid (q_2 \mid q_3)$
$(q_1 \mid q_2)$ **where** q_3 is equivalent to $q_1 \mid (q_2$ **where** $q_3)$
$(q_1 \mid q_2).q_3$ is equivalent to $q_1.(q_2.q_3)$

q_1 **where** (q_2 **and** q_3) is equivalent to (q_1 **where** q_2) **where** q_3

Proofs follow from the distributivity property that holds for these operators and from the analysis of the contents of ES sections that are opened by these operators. Additional (rather weak) assumptions are sometimes necessary. There are many other rewriting rules that can be derived from the distributivity property.

The above properties increase a lot the power of our method based on detecting and factoring out independent subqueries. For example, let's analyze the query returning lectures along with professors who conduct them, provided that those professors are older than forty and credits of those lectures are higher than 5:

Lecture | (*conducted_by*.(*Professor* **where** *age* > 40 **and** *credit* > 5)) **(3)**
(1,1) 2 (2,2) 3 (3,3) 4 (4,4) (4,2)

Name *credit* in the predicate *credit* > 5 is bound in section 2, though the stack size is 4, hence the predicate is independent of two non-algebraic operators. The subquery *credit* > 5 is combined with the rest of the selection predicate by **and**. Due to the distributivity property the condition *credit* > 5 can be pushed and applied to the left subquery of the join operator (on which it is dependent), that is, to the subquery *Lecture*. Therefore the query (3) can be rewritten to the following form:

(*Lecture* **where** *credit* > 5) | (*conducted_by*.(*Professor* **where** *age* > 40))

The example shows pushing a predicate before three different non-algebraic operators (**where**, dot and |).

7 Conclusions and Future Work

We have presented a formal stack-based model, which allows us to describe precisely the semantics of OQL-like query languages. In this setting we have described a general query optimization method based on detecting and factoring out independent subqueries. In comparison to similar proposals, our method is much more general. The reason is the fact that in our approach the semantics of all non-algebraic operators (**where**, dot, join, quantifiers, etc.) is the same, except for the construction of the final result. Our method is also independent of the complexity of a subquery to be factored out. More powerful variants of the method we have received due to the distributivity property of **where**, dot and join operators.

We see many extensions of the presented methods. Currently we are working on other kinds of rewriting rules: removing dead subparts of a query, changing joins into dots, reducing auxiliary names, transformations involving indices, and others.

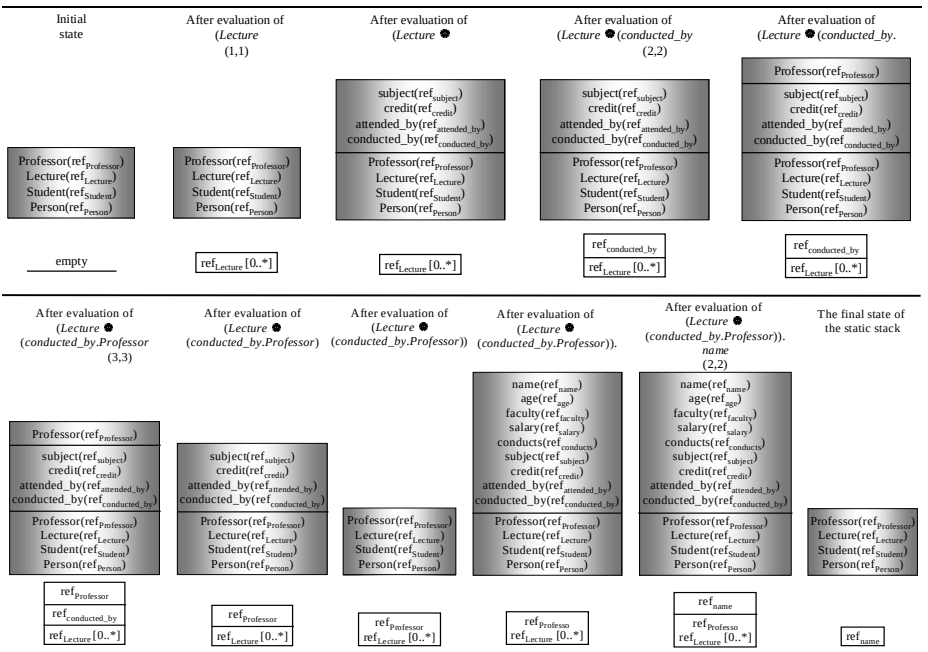
References

1. Christophides, V., Abiteboul, S., Cluet, S., Moerkotte, G.: Evaluating Queries with Generalized Path Expressions. Proc. of SIGMOD Conf., 413-422, 1996
2. Cluet, S., Delobel, C.: Towards a Unification of Rewrite-based Optimization Techniques for Object-Oriented Queries. (In) Query Processing for Advanced Database Systems. Morgan Kaufmann, 245-272, 1994

3. Graefe, G.: Query Evaluation Techniques for Large Databases. ACM Computing Surveys 25(2), 1993
4. Grust, T., Kröger, J., Gluche, D., Heuer, A., Scholl, M.H. Query Evaluation in CROQUE - Calculus and Algebra Coincide. Proc. of 15th British National Conf. on Databases, Springer LNCS 1271, 84-100, 1997
5. Ioannidis, Y.E.: Query Optimization. Computing Surveys 28(1), 121-123, 1996
6. ODMG Team: The Object Database Standard ODMG, Release 2.0., Morgan Kaufman, 1997
7. Rupp, S.: A Critique of Release 2.0 of ODMG-93 Standard. http://gameboy.gia.rwth-aachen.de/odmgbugs
8. Subieta, K., Kambayashi, Y., Leszczyłowski, J.: Procedures in Object-Oriented Query Languages. Proc. of VLDB Conf., 182-193, 1995
9. Subieta, K.: LOQIS: The Object-Oriented Database Programming System. Proc. 1st Intl. East/West Database Workshop on Next Generation Information System Technology, Springer LNCS 504, 403-421, 1991
10. Subieta, K., et al.: A Critique of Object Algebras. Unpublished, Nov.1995, http://www.ipipan.waw.pl/~subieta/EngPapers/index.html
11. Subieta, K.: Object-Oriented Standards: Can ODMG OQL be Extended to a Programming Language? Proc. of CODAS Symp., World Scientific, 459-468, 1997
12. Tsimmis Team: The TSIMMIS Project, 1994-95; http://www-db.stanford.edu/pub

Appendix: Static Query Analysis

We present analysis of the query: (*Lecture* | (*conducted_by.Professor*)).*name*
In the figures we present the static ES (gray), then below the static QRES (white). For each name being bound we show a pair denoting the stack size and the binding level.



Compositional Specification Calculus for Information Systems Development

Leonid Kalinichenko

Institute for Problems of Informatics,
Russian Academy of Sciences,
Vavilov str., 30/6, Moscow, V-334, 117900
leonidk@synth.ipi.ac.ru

Abstract. The paper presents a novel approach for type specification manipulations as the basic operations intended to develop various forms of compositions in information systems. Among them are interoperable compositions of pre-existing components formed during the information systems design, heterogeneous multidatabase compositions, database schema and ontology integration, compositions of workflows, compositions of the type of the result of algebraic operations over typed data collections. In the paper the compositional specification calculus is presented in context of one of such application – the compositional information systems development. Principle of decomposition of type specifications into a collection of type reducts serving as units of reuse and composition is formulated. An operation of taking most common reduct for component and specification of requirements types is defined. The refinement property of the common reduct leads to a justifiable identification of reusable component fragments. Type lattice and algebra based on partially ordered set of complete type specifications and the type commonality principle are defined. Type specification composition operations of the algebra are based on identification of common information in the composed specifications. Paper presents a combination of semi-formal and formal modeling facilities to perform provably correct operations of common reduct identification, type specification composition and reuse.

Keywords: compositional specification calculus, type compositions, most common reducts, type refinement, type lattice, compositional information systems development

1 Introduction

In this paper we address a novel approach for composition-oriented specification manipulation that has been elaborated in frame of the SYNTHESIS method.

The overall goal of the SYNTHESIS project [1] [7,10,9,5] is to provide a uniform collection of modeling facilities and methods suitable for different forms

[1] The SYNTHESIS project being developed at the Institute for Problems of Informatics of the Russian Academy of Sciences (IPI RAS) has been partially supported by the INTAS grant 94-1817, INTAS-OPEN grant 97-1109 and the Russian Foundation for Basic Research grant 97-07-90369

H. Krawczyk (Ed.): CRYPTO'98, LNCS 1462, pp. 317–331, 1998.
© Springer-Verlag Berlin Heidelberg 1998

of compositions applicable for information systems. Examples of various compositions include interoperable compositions of pre-existing components formed during the information systems development, heterogeneous multidatabase compositions, database schema and ontology integration, compositions of workflows, compositions of the type of the result of algebraic operations over typed data collections in object or object-relational databases. The SYNTHESIS objective is to work out general solutions for various forms of compositions supporting them with suitable methods and tools.

Presenting general ideas of the compositional specification calculus, the paper will be focused on one specific form of its application – the compositional information systems development (CISD) where the calculus have been actively used. [2] Different phases of CISD should support forward engineering activities (that lead to construction of new information systems with reuse of pre-existing components) as well as reverse engineering activities (intended for extraction of component specifications from existing (legacy) information system descriptions and equivalently presenting them in canonic and complete way for further reuse). Different aspects of the SYNTHESIS CISD approach were considered in [8,4,10,9,5].

The SYNTHESIS CISD method belongs to *formal and knowledge-based techniques* focusing on component composition process based on the combination of formal and semi-formal facilities. For specifications (of requirements or components) precise semantics are required to rely on them in process of compositions. Application semantics and object model semantics are distinguished. The former is defined by ontological specifications and the latter is represented in frame of the "canonical" semi-formal object model used for uniform representations of various object and data model component specifications in one paradigm. To give the canonical model exact meaning, a mapping of this object model into the formal one (currently this is the Abstract Machine Notation (AMN) of the B-Technology [1,2]) has been constructed.

Specifying components developed in different object models, we should preserve information and operations of their definitions while mapping them into the respective canonical entities. The required state-based and behavioural properties of the mappings lead to a proof that an original, source component type model is a refinement of its mapping into the canonical type model [10]. Correct compositions of information systems components should be semantically interoperable in contexts of specific applications. Application semantics and ontological modeling are beyond this paper, more on that can be found in [4,5].

According to SYNTHESIS, steps of the CISD process include integration of application domain and information resource ontological contexts establishing ontological relevance of constituents of the requirements and components specifications, identification of component types (classes) and their fragments suitable for the concretization of an analysis model type (class) capturing its structural,

[2] In this paper we consider conventional type specifications. Types exhibiting interactive behaviour conform to the same methodology but require additional techniques for the specification calculus foundations [11].

extensional and behavioural properties, composition of such fragments into specifications concretizing the requirements, justification of a property of concretization of requirements by such compositions [5].

The design is a process of systematic manipulation and transformation of specifications. Type specifications of the canonical model are chosen as the basic units for such manipulation. The manipulations required include decomposition of type specifications into consistent fragments, identification of reusable fragments (patterns of reuse), composition of identified fragments into specifications concretizing the requirements, justification of reusability and substitutability of the results of such transformations instead of the specifications of requirements.

The paper presents the compositional specification calculus intentionally designed for such manipulations. According to the approach, compositions of component specifications are based on common fragments discovered and on the refinement technique. This approach significantly differs from techniques considering compositions by interconnecting components on contractual basis (e.g., by scripting languages) [13].

The paper is structured as follows. An overview of the canonical modeling facilities providing for complete specification of components and requirements is presented in the second section. The third section introduces compositional specification calculus including common reduct idea, type composition operations and their properties. Type specification composition example using canonical model notation is presented in the fourth section. A technique applying formal notation to prove properties of type specification compositions is briefly considered further. Related works are briefly discussed in the last section. Conclusion summarizes the results.

2 Overview of the basic features of the canonical model

The Ontological model, Requirement planning/Analysis model, Design model, Implementation model, Component (Information Resource) Specification model and respective macro layers are constituents of the SYNTHESIS CISD framework [5]. The semantics behind any of these models is provided by one and the same descriptive canonical object model treated in a semi-formal style and having a formal interpretation. The canonical model should provide for the integrated support of various functions, including (i) semi-formal representation of specification of requirements and analysis models of information systems; (ii) description of ontological contexts of application domains and justification of their coherence; (iii) uniform (canonical) representation of specifications of heterogeneous components; (iv) support of semantic reconciliation and adaptation of components to form their reducts and compositions serving as concretizations of analysis models of information systems.

To cope with that, the semi-formal canonical model should have a formal interpretation. Mapping of semi-formal specification into a formal one allows to develop a formal model of an application domain and components reflecting their static and dynamic properties. Formal specifications and respective tools

create a basis for provable reasoning on the specification consistency and provable concretization of specification of requirements by pre-existing components compositions.

The SYNTHESIS language [8] defines specification-oriented semi-formal canonical model. Here we present the very basic canonical language features to make the examples demonstrating ideas of the Compositional Specification Calculus readable. It is important to note that the Specification Calculus considered does not depend on any specific notation or modeling facilities. Canonical model provides support of wide range of data - from untyped data on one end of the range to strictly typed data on another. Untyped data are represented as *frames* that are used as symbolic models of some entities or concepts. The language uses frames to describe any entity, including the entities of the language itself, such as types, classes, functions, assertions. A frame at any moment of its life cycle can be declared to belong to an admissible class (class is a collection of typed objects). At that moment the frame becomes an *object.*

Typed data should conform to *abstract data types* (ADT) prescribing behaviour of their instances by means of the type's operations. ADT describes the syntactic interface of the type (its signature) and operands whose signatures define names and types of their operations and their specifications define the operation algorithms.

Type specifications are syntactically represented by frames, their attributes - by slots of the frames. Syntactically frames are inserted into figure brackets { and }, slots are represented as pairs <slot name > : <slot value> (a frame can be used as a slot value), slots in a frame are separated by semi-colons.

Each frame may be declared to belong to one or several classes (metaclasses). Such frame membership is denoted by a slot in :< class or metaclass name list >.

Syntactically, each functional (non-state) attribute of a type is defined by description of a function:

< function description >::=< function identifier >; in : function;
[params : {< formal parameter list >};][< specification >]}
< formal parameter identifier >::=< parameter sort symbol >< typed variable >
< parameter sort symbol >::= −| + | < empty >

The meaning of a parameter sort symbol is:

'+' - input parameter; '-' - output parameter; < empty > - input & output parameter.

A function is defined by a predicative specification stating its mixed pre/post conditions. To describe a state transition, it is necessary to distinctly denote the variables that define the state before and after the function execution. Variables referring to the state after function application are primed.

Formulae appearing in the predicative specifications below are described as follows. A variant of the multisorted first order predicate logic language is used in SYNTHESIS [8]. Each predicate, function, variable and constant in the formulae is typed. In simplified form formulae are either *atoms* or appear as:

w_1 & w_2 (w_1 and w_2)
w_1 | w_2 (w_1 or w_2)
$\neg w_2$ (not w_2)
$w_1 -> w_2$ (if w_1 then w_2)
$\exists$ x/t (w) (for some x of type t, w)
$\forall$ x/t (w) (for all x of type t, w)

where w, w_1, w_2 are formulae. Existential and universal quantifiers are denoted by ex and all, respectively.

3 Compositional specification calculus

3.1 Subtyping relation and type reducts

A signature Σ_T of a type specification $T =< V_T, O_T, I_T >$ includes a set of operation symbols O_T indicating operations argument and result types and a set of predicate symbols I_T (for the type invariants) indicating predicate argument types. Conjunction of all invariants in I_T constitutes the type invariant. We model an extension V_T of each type T (a carrier of the type) by a set of proxies representing respective instances of the type.

Among the type T operations we shall distinguish state attributes $Att_T \subseteq O_T$. Each state attribute is modelled as a function $V_T \rightarrow V_S$ where V_S denotes a carrier of the attribute type.

We assume the existence for our types of an abstract interpretation universe $\mathcal{V}$ (that is assumed to consist of all proxies of instances of any type). A type is a specification (intensional denotation) of a subset of elements in this universe. In accordance with the denotational semantics concepts, $\top$ is the *least informative* type denoting the whole universe $\mathcal{V}$ and $\bot$ is the *overdefined,* inconsistent type denoting the empty set. The subtype relation is the *information ordering* relation $\preceq$. Extensional interpretation of $\preceq$ is by the set inclusion. A subtype is considered to be more informative than its supertype (we assume a subtype relation definition based on type specifications similar to the given in [12]). Let $\mathcal{T}$ be a set of types with a subtype ordering relation $\preceq$. An extensional type semantics is an order homomorphism [3]:

$h : (\mathcal{T}, \preceq) \rightarrow (\mathbf{P}\mathcal{V}, \subseteq)$
In particular:
$h(\top) = \mathcal{V}, h(\bot) = \{\}$
and for all T, S in $\mathcal{T}$
$S \preceq T \Rightarrow h(S) \subseteq h(T)$

These considerations give us a general hierarchy for placement of type specifications. The guiding intuition behind the hierarchy is that type specifications are partial descriptions of real-world entities ordered by the information content.

h(T) gives a set of proxies V_T. Each proxy has interpretation in a certain domain D that also is a poset: $D, \preceq$. The bottom element $\bot_D$ is assumed to be defined for each instance domain such that for any $d \in D$, $\bot_D \preceq d$.

The domains (sets of type instance state values) are defined using complex sort constructors like cartesian product ($\times$), powerset ($\mathbf{P}$), set comprehension

$(\{x|x \in s \wedge P\})$, relational sort constructors $(s \leftrightarrow t)$, functional sort constructors $(s \longrightarrow t)$, etc.

Definition 1. Type reduct *A signature reduct R_T of a type T is defined as a subsignature Σ'_T of type signature Σ_T that includes a carrier V_T, a set of symbols of operations $O'_T \subseteq O_T$, a set of symbols of invariants $I'_T \subseteq I_T$.*

This definition from the signature level can be easily extended to the specification level so that a type reduct R_T can be considered a *subspecification* (with a signature Σ'_T) of specification of the type T. The specification of R_T should be formed so that R_T becomes a supertype of T. We assume that only the states admissible for a type remain to be admissible for a reduct of the type (no other reduct states are admissible). Therefore, the carrier of a reduct is assumed to be equal to the carrier of its type.

Operations and invariants of the reduct subspecifications taken from the original type specification should be systematically modified. The occurences of the discarded attributes of the original type into the operations and invariants of the reduct should be existentially quantified and properly ranged. For the type T and its reduct R_T the formula is formed as follows:

> $\exists t/T(r = t/R_T$ & <predicate with discarded attributes; each of the latter should be qualified by $T >$)

Here the typed variable r/R_T is assumed to be universally quantified.
In the sequel, speaking of reducts, we have in mind the specification reducts.

3.2 Type refinement

Using the operation ρ of taking a reduct $R_T = \rho(T, O_r)$ of type $T \in \mathcal{T}$ a set of type reducts $\{R_T \mid T \preceq R_T \wedge O_r \subseteq O_T\}$ can be produced. Thus, decomposing a type specification, we can get its different reducts on the basis of various type specification subsignatures.

Taking into account that types we consider are characterized by their state and behaviour, we introduce definition of type refinement as follows.

Definition 2. *Type U is a* refinement *of type T iff*

- *there exists a one-to-one correspondence $Ops : O_T \Leftrightarrow O_U$;*
- *there exists an abstraction function $Abs : V_T \rightarrow V_U$ that maps each admissible state of T into the respective state of U;*
- $\forall x \in V_T \exists y \in V_U(Abs(x, y) \Rightarrow I_T \wedge I_U)$
- *for every operation $o \in O_T$ the operation $Ops(o) = o' \in O_U$ is a refinement of o. To establish an operation refinement it is required that operation precondition $pre(o)$ should* imply *the precondition $pre(o')$ and operation postcondition $post(o')$ should* imply *postcondition $post(o)$.*

3.3 Common reducts

Based on the notions of reduct and type refinement, a measure of common information between types in $\mathcal{T}$ can be established.

Definition 3. *A* common reduct *for types T_1, T_2 is such reduct R_{T_1} of T_1 that there exists a reduct R_{T_2} of T_2 such that R_{T_2} is a refinement of R_{T_1}. Further we refer to R_{T_2} as to a* conjugate *of the common reduct.*

Though taking a common reduct for two types is not tractable, the following simple heuristics show what can be reasonable approaches for that. Operations of the type T_1 are suspected to belong to its common reduct with T_2 if operations with the equal signatures can be found in T_2 modulo variable renaming, parameters ordering and parameter type redefinition (contravariant for the argument types and covariant for the result types type differences for T_2 are acceptable). A specification of an operation of T_1 to be included into the resulting reduct is chosen among such pre-selected pairs of operations of operand types if the operations in a pair are in a refinement order (for the common reduct (resulting supertype) more abstract operation should be chosen). If the pre-selected operations are not in a proper refinement order then they are considered to be different operations and will not be included into the common reduct.

Definition 4. *A* most common reduct *$R_{MC}(T_1, T_2)$ for types T_1, T_2 is a reduct R_{T_1} of T_1 such that there exists a reduct R_{T_2} of T_2 that refines R_{T_1} and there can be no other reduct $R^i_{T_1}$ such that $R_{MC}(T_1, T_2)$ is a reduct of $R^i_{T_1}$, $R^i_{T_1}$ is not equal to $R_{MC}(T_1, T_2)$ and there exists a reduct $R^i_{T_2}$ of T_2 that refines $R^i_{T_1}$.*

Reducts provide for type specification decompositions thus creating a basis for their further compositions.

3.4 Type compositions

We introduce here type composition operations that can be used to infer new types from the existing ones.

Let $T_i (1 \leq i \leq n) \in \mathcal{T}$ denotes types.

Definition 5. Type meet *operation. An operation $T_1 \sqcap T_2$ produces a type T as an 'intersection' of specifications of the operand types. Generally the result T of the meet operation is formed as the merge of two most common reducts of types T_1 and T_2 : $R_{MC}(T_1, T_2)$ and $R_{MC}(T_2, T_1)$. The merge of two reducts includes union of sets of their operation specifications. If in the union we get a pair of operations that are in a refinement order then only one of them, the more abstract one is included into the merge. Invariants created in the resulting type are formed by* disjuncting *invariants taken from the most common reducts specifications being merged.*

If T_2 (T_1) is a subtype of T_1 (T_2) then T_1 (T_2) is a result of the meet operation. Type T is placed in the type hierarchy as an immediate supertype of the meet arguments types and a direct subtype of all common direct supertypes of the meet argument types.

Meet operation produces a type T that contains common information contained in types T_1 and T_2.

Definition 6. Type join *operation. An operation* $T_1 \sqcup T_2$ *produces type* T *as a 'join' of specifications of the operand types. Generally* T *includes a merge of specifications of* T_1 *and* T_2. *Common elements of specifications of* T_1 *and* T_2 *are included into the merge (resulting type) only once. The common elements are determined by another merge - the merge of conjugates of two most common reducts of types* T_1 *and* T_2 : $R_{MC}(T_1, T_2)$ *and* $R_{MC}(T_2, T_1)$. *The merge of two conjugates includes union of sets of their operation specifications. If in the union we get a pair of operations that are in a refinement order then only one of them, the more refined one (belonging to the conjugate of the most common reduct) is included into the merge. Invariants created in the resulting type are formed by* conjuncting *invariants taken from the original types.*

If $T_2(T_1)$ *is a subtype of* $T_1(T_2)$ *then* $T_2(T_1)$ *is a result of a join operation. A type* T *is placed in the type hierarchy as an immediate subtype of the join operand types and a direct supertype of all the common direct subtypes of the join argument types.*

3.5 Type lattice and properties of compositions

Partially ordered by a subtyping relation set of types $\mathcal{T}$ with $\top$ and $\bot$ types is a *lattice:* for all $S, U \in \mathcal{T}$ there exists least upper bound (l.u.b.) and greatest lower bound (g.l.b.). Meet $T = S \sqcap U$ produces T as the g.l.b. for types S, U in the $(\mathcal{T}, \preceq)$ lattice (in case when S, U have no common reducts, meet results in $\top$). Join $T = S \sqcup U$ produces T as the l.u.b. for types S, U in the $(\mathcal{T}, \preceq)$ lattice (in case when S, U have no common reducts, join results in $\bot$).

$< \mathcal{T}; \sqcap, \sqcup >$ is an algebra with two binary operations. For such algebra the following properties are established:

1. Commutativity: $S \sqcap U = U \sqcap S$ and $S \sqcup U = U \sqcup S$
2. Associativity: $S \sqcap (U \sqcap S) = (S \sqcap U) \sqcap S$ and $S \sqcup (U \sqcup T) = (S \sqcup U) \sqcup T$
3. Idempotence: $S \sqcap S = S$ and $S \sqcup S = S$
4. Absorption: $S \sqcap (S \sqcup U) = S$ and $S \sqcup (S \sqcap U) = S$

Two another rules relate subtyping with operations meet and join:
$S \sqcap U = U \equiv S \preceq U$
$S \sqcup U = S \equiv S \preceq U$
as well as:
$S \preceq S \sqcap U$ and $U \preceq S \sqcap U$;
$S \sqcup U \preceq S$ and $S \sqcup U \preceq U$.

4 Type composition example

For a funding agency supporting projects we assume the following. We distinguish between research projects (*Rproject* type) and industrial projects (*Iproject*

type). Industrial projects can have additional support from a sponsor company. Any organization (university or company) can be a coordinator of a research project. Only companies can coordinate industrial projects. Projects can cooperate with other projects: research projects – only with research projects and industrial projects – with any kind of projects. A project can be nominated as a candidate for a cooperation with a given project by a function *candidate_proj*. Research and industrial projects for computer science have additional constraints. We specify the example using the canonical model that briefly has been already introduced:

Type declarations:

```
{Project;
 in: type;
 area: string;
 grade: real;
};

{Rproject;
 in: type;
 supertype: Project;
 coordinator: Organization;
 leader: Professor;
 priority_theme: string;
 cooperate_with: {set_of: Rproject};

 candidate_proj: {in: function;
 params: { j/Rproject, -c/Project};
 {{this.area = j.area & this.priority_theme =
 j.priority_theme & c' = j}}};

 area_constr: {in: predicate, invariant;
 {{ all p/Rproject (p.area = 'comp-sci' => p.grade = 5 &
 (p.priority_theme = 'open systems' |
 p.priority_theme = 'interoperability'))}}};

 leader_constr: {in: predicate, invariant;
 {{ all p/Rproject (p.leader.degree = 'PhD') }}}
 }

{Iproject;
 in: type;
 supertype: Project;
 coordinator: Company;
 cooperate_with: {set_of: Project};
 sponsor: Company;
```

```
 candidate_proj: {in: function;
 params: {j/Project, -c/Project};
 {{this.area = j.area & c' = j}}};

 area_constr: {in: predicate, invariant;
 {{ all p/Iproject (p.area = 'comp-sci' => p.grade = 5 )}}}
 }
```

Now we show how to get type compositions for the types *Rproject* and *Iproject.*

4.1 Getting most common reducts for the types to be composed

To get a meet type of the types *Rproject* and *Iproject* we should define the most common reducts of types *Rproject* and *Iproject:* $R_{MC}(Rproject, Iproject)$ and $R_{MC}(Iproject, Rproject)$.

The $R_{MC}(Rproject, Iproject)$ looks as follows:

```
{RIrmc;
 in: type;
 supertype: Project;
 coordinator: Organization;

 candidate_proj: {in: function;
 params: {j/Rproject, -c/Project};
 {{this.area = j.area &
 ex p/Rproject (this = p/RIrmc &
 p.priority_theme = j.priority_theme) &
 c' = j}}};

 area_constr: {in: predicate, invariant;
 {{ all r/RIrmc (r.area = 'comp-sci' => r.grade = 5 &
 ex p/Rproject (r = p/RIrmc &
 (p.priority_theme = 'open systems' |
 p.priority_theme = 'interoperability')))}}};

 leader_constr: {in: predicate, invariant;
 {{ all r/RIrmc
 ex p/Rproject (r = p/RIrmc &
 p.leader.degree = 'PhD')}}}
 }
```

The $R_{MC}(Iproject, Rproject)$ looks as follows:

```
{IRrmc;
 in: type;
 supertype: Project;
```

```
cooperate_with: {set_of: Project};

area_constr: {in: predicate, invariant;
{{ all r/IRrmc (r.area = 'comp-sci' => r.grade = 5)}}
}
```

4.2 Composition meet (Rproject, Iproject)

The type *RImeet* – result of the meet composition – becomes a supertype of types *Rproject* and *Iproject*.

```
{RImeet;
 in: type;
 supertype: Project;
 coordinator: Organization;
 cooperate_with: {set_of: Project};

 candidate_proj: {in: function;
 params: {j/Rproject, -c/Project};
 {{this.area = j.area &
 ex p/Rproject (this = p &
 p.priority_theme = j.priority_theme) &
 c' = j}}};

 area_constr: {in: predicate, invariant;
 {{ all r/RImeet (r.area = 'comp-sci' => r.grade = 5 )}};

 leader_constr: {in: predicate, invariant;
 {{ all r/RImeet
 ex p/Rproject (r = p/RImeet &
 p.leader.degree = 'PhD') }}}
 }
```

The type *RIjoin* – result of the join composition – can be constructed similarly using conjugates of the common reducts defined above.

5 Proving of the common reducts and compositions properties

A technique of establishing the required properties of compositions consists in mapping of type specifications into a formal notation and proving consistency and refinement of the specifications. Abstract Machine Notation (AMN) is applied for that. Practical importance of establishing provable properties of type specifications has been shown in [6].

Principles of mapping of canonical model specifications into AMN consist in the following. For simplictity, we assume here that each canonical model type

specification is mapped into a separate abstract machine. Sets of proxies related to data types are interpreted in AMN as finite sets related in accordance with the subtype relationship so that $V_{T_{sub}} \subseteq V_{T_{sup}}$ for each pair of subtype T_{sub} and the respective supertype $\mathsf{T_{sup}}$ declared in SYNTHESIS. Such constant sets provide a natural basis to model a type specification by a collection of functions of AMN. Each such function corresponds to a state attribute of the type. Specifications of sets denoting extents of types are collected in the contextual abstract machine *Ctxt.mch*.

Based on the extensional principle, we model state attributes of the SYNTHESIS type T in the **invariant** clause of abstract machine interpreting attributes as functions $attr \in A_T \rightarrow A_{T_a}$ where A_{T_a} is an extent of the *attr* value of the T_a type. A kind of the function depends on the properties of the state attribute.

State attributes are represented also by an operation *get_attr*, and for mutable attributes additionally by an operation *set_attr* in the **operations** clause of an abstract machine.

The extensional constants will be widely used also in the SYNTHESIS specifications for proper typing of variables in invariants and operations of abstract machines. For instance, to pass a value of a type as a parameter of an operation it is sufficient to type this variable with the respective extent set.

SYNTHESIS formulae used in invariants are mapped into the AMN predicates. SYNTHESIS formulae used in specifications of functions are mapped into the AMN substitutions.

Based on such mapping, we claim:

Proposition. Type U is a refinement of type T iff their respective mappings to AMN U_a and T_a are such that U_a refines T_a.

Space limitation does not allow to include into the paper the AMN specifications corresponding to the example presented in the previous section. To prove the common reduct properties it is required (i) to map the common reduct for *Rproject* and *Iproject* types into an abstract machine; (ii) to map the conjugate of the comon reduct for *Rproject, Iproject* types into a refinement of this machine; (iii) to justify the refinement properties proving obligations of the refinement machine generated by B-Toolkit. Similarly properties of specification compositions can be proved.

6 Related works

Research on type lattices and respective algebras has already quite long history. Having different motivations, the respective developments may be characterized as a compromise between reasonable expressiveness and tractability of the modeling facilities. Our paper emphasizes expressiveness sacrifizing tractability and considers complete type specifications. Such decision is motivated by orientation of the modeling facilities proposed at the process of composition. The benefits we get include rigorous way of identification of common fragments of type specifications leading to their justifiable compositions for reuse.

H. Ait-Kaci was one of the first who introduced a syntactic calculus of record-like structures based on a type subsumption ordering forming a lattice structure. Structural properties of types were taken into account. Solving of systems of type equations by iterated rewriting of type symbols became possible [3].

A.Ohori continued investigation of structural type systems (including labeled records, labeled disjoint unions, finite sets and recursion) as a proper generalization of the relational data model. Tractable type inference has been emphasized [15].

R.J.Peters introduced an object model with meet, join and product operations for type signatures intended for query type signature inference in course of object database access [16].

R.Mili, A.Mili, R.Mittermeier considered structure of component repository as an information retrieval system. Component retrieval problem is considered for functions as components modeled with relations containing all admissible pairs for the function input/output. Refinement ordering of functions has lattice properties. The lattice is formed by join and meet operations on relations representing functions. Join (meet) represents the sum of (common) specification information that is contained in both relations [14].

A. Zaremski and J. Wing proposed signature matching as a mechanism for retrieving software components from a component repository. Queries and components are represented by their signatures. Further they considered specification matching for functions represented by their pre- and post- conditions [17].

7 Conclusion

Type specification manipulations for compositional information systems development require specific operations that could lead to the justifiable 'calculation' of the required concretizing specifications. Complete specifications sufficient for provable identification of reusable fragments are assumed in the CISD context for components as well as for the requirements.

A novel approach for type specification manipulation for various compositions during information systems development is proposed. The approach has the following distinguishing features:

1. Decomposition of type specifications into a collection of type reducts serving as units of reuse and composition;
2. Incorporation of taking common reduct operation for component and specification of requirements types used for discovery of reusable specification fragments;
3. Application of type specification refinement as a fundamental propery for justification of type commonality and correctness of type specification fragments substitution;
4. Introduction of composition operations intended for complete type specifications and based on identification of common information in the composed specifications;

5. Definition of type lattice and algebra based on partially ordered set of complete type specifications and common reduct idea;
6. Applying a combination of semi-formal and formal modeling facilities to make provably correct the operations of common reduct identification, type specification composition and reuse.

Compositional specification calculus based on this approach has been introduced. One of the natural application of the approach is CISD in a distributed object middleware environment (like CORBA) where resulting specifications can be implemented as composite objects constructed by interoperation of reusable fragments applying adapters (wrappers) above original components. The tool supporting the CISD approach had been developed as complementing existing Object Analysis and Design facilities to make them truly component-based [5].

The compositional specification calculus developed is intended for different forms of compositions applicable for information systems. Besides CISD, this might be heterogeneous multidatabase compositions, database schema and ontology integration, compositions of workflows, composition of the type of the result of algebraic operations over typed data collections. The approach presented is being extended now for interactive type specifications suitable for modeling such artifacts as workflows [9,11].

References

1. Abrial J.-R. *The B Book: assigning programs to meaning,* Cambridge University Press, 1996 318
2. Abrial J.-R. B-Technology. Technical overview. BP International Ltd., 1992, 73 p. 318
3. Ait-Kaci H. An algebraic semantic approach to the effective resolution of types equations. Theoretical computer science, 45, 1986, 293 - 351 321, 329
4. Briukhov D.O., Shumilov S.S. Ontology Specification and Integration Facilities in a Semantic Interoperation Framework, In *Proc. of the International Workshop ADBIS'95,* Springer, 1995 318, 318
5. Briukhov D., Kalinichenko L. Component-based information systems development tool supporting the SYNTHESIS design method. Springer LNCS, *Proceedings of the East European Symposium on "Advances in Databases and Information Systems",* September 1998, Poland 317, 318, 318, 319, 319, 330
6. Jezequel J.-M., Meyer B. Design by Contract: The Lessons of Ariane, http://www.tools.com/doc/manuals/technology/contract/ariane/index.html 327
7. Kalinichenko L.A. Emerging semantic-based interoperable information system technology. In *Proceedings of the International Conference Computers as our better partners*, Tokyo, March 1994, World Scientific 317
8. Kalinichenko L.A. SYNTHESIS: the language for desription, design and programming of the heterogeneous interoperable information resource environment. Institute for Problems of Informatics, Russian Academy of Sciences, Moscow, 1995 318, 320, 320
9. Kalinichenko L.A. Workflow Reuse and Semantic Interoperation Issues. In *Advances in workflow management systems and interoperability.* A.Dogac, L.Kalinichenko, M.T. Ozsu, A.Sheth (Eds.). NATO Advanced Study Institute, Istanbul, August 1997 317, 318, 330

10. Kalinichenko L.A. Method for data models integration in the common paradigm. In *Proceedings of the First East European Workshop 'Advances in Databases and Information Systems'*, St. Petersburg, September 1997 317, 318, 318
11. Kalinichenko L.A. Composition of type specifications exhibiting the interactive behaviour. In *Proceedings of EDBT'98 Workshop on Workflow Management Systems,* March 1998, Valencia 318, 330
12. Liskov B., Wing J.M. Specifications and their use in defining subtypes. Proceedings of OOPSLA 1993, ACM SIGPLAN Notices, vol. 28, N 10, October 1993 321
13. Lumpe M. A Pi-Calculus Based Approach to Software Composition, Ph.D. thesis, University of Bern, Institute of Computer Science and Applied Mathematics, January 1999 319
14. Mili R., Mili A., Mittermeir R. Storing and retrieving software components: a refinement based systems. IEEE Transactions on Software Engineering, v. 23, N 7, July 1997 329
15. Ohori A. Semantics of types for database objects. Theoretical Computer Science, 76, 1990, p. 53 - 91 329
16. Peters R.J. TIGUKAT: a uniform behavioral objectbase management system. Technical report TR 94-06, April 1994, University of Alberta 329
17. Zaremski A.M., Wing J.M. Specification matching of software components. ACM Transactions on Software Engineering and Methodology, v. 6, N 4, October 1997 329

The Impact of Using Class Inheritance in a Distributed Information System

Patrick Dunne[1] and Alex Gray [2]

[1] BT Laboratories, Martlesham, Ipswich, UK
pjd6@tutor.open.ac.uk
[2] University of Wales, College of Cardiff, Cardiff, UK
w.a.gray@cf.cs.ac.uk

Abstract. The use of class inheritance provides many benefits when developing information based systems. The use of class inheritance within a distributed object information system (DOIS) however, has failed to gain acceptance within the distributed community. One of the major reasons for this is the general acceptance of the widely held belief [19] that inheritance is said to unacceptably degrade the performance of a DOIS. This widely held belief arose from the analysis of Smalltalk and its use in a distributed system. We argue that newer object-oriented languages such as Eiffel and Java use more efficient mechanisms to implement object-oriented features including inheritance. Also, more efficient mechanisms exist for supporting distribution which have changed the distributed inheritance situation. The development of the network techniques such as RPC, CORBA and RMI provide improved support for DOIS. This paper presents an examination of the use of CORBA, RMI and RPC with regard to the performance overheads that arise from moving data from one machine to another. The analysis examines the additional effort that is required in order to support distributed class inheritance. Each aspect of distribution, such as marshalling, binding, network latency and inter-process communication was measured in order to establish the reasons for such degradation. Our results show that the poor performance which is associated with the use of class inheritance within a DOIS, is mainly a result of the inefficiency of the support provided by a language and the unacceptable degradation is not a direct consequence of distribution. Furthermore, our study shows that network techniques such as CORBA and RMI although providing a high level of abstraction, are generally not efficient when large amounts of data are being sent across the network.

1. Introduction

Currently, the inheritance mechanisms that are provided by single-user information systems tend to be based on class inheritance, which provides such benefits as incremental development, specialisation of behaviour and behaviour sharing. Inheritance mechanisms that are found on DOIS systems are quite different and usually constrain the way in which inheritance can be applied. These constraints were imposed to avoid problems such as the inheritance anomaly [11], and the missing base class [17] which are more evident in distributed systems. As concurrency is a

Eder et al. (Eds.): ADBIS´99, LNCS 1691, pp.332 -348, 1999.
© Springer-Verlag Berlin Heidelberg 1999

feature of distributed systems, in the case of inheritance, it has been found that in certain circumstances, concurrency conflicts with the use of class inheritance.

By having different inheritance mechanisms for use in a single-user and distributed system, this has resulted in different object models being used for information systems that are to be single-user based compared to an information system that is to be used with a distributed system. DOIS generally uses object inheritance [5] to enable the sharing of behaviour between objects that reside in a distributed system. This is a more constrained form of inheritance compared to class inheritance, where class inheritance is commonly used on single-user systems to support behaviour sharing. The use of class inheritance provides a higher level of reuse compared to object inheritance, as it enables the customisation of classes as well as the sharing of behaviour. One of the main criticisms of using class inheritance in a distributed system is the unacceptable performance degradation that results from its use. Previous studies [2], analysed various implementation methods in an effort to provide an efficient implementation mechanism for class inheritance within the Smalltalk language. The results of this work concluded that class inheritance was too inefficient for distributed systems. The main cause of this level of degradation was attributed to the dynamic binding features of class inheritance. Consider the hierarchy in fig. 1, where class A inherits class B and class B inherits class C, and each of the classes resides on different workstations within a distributed system.

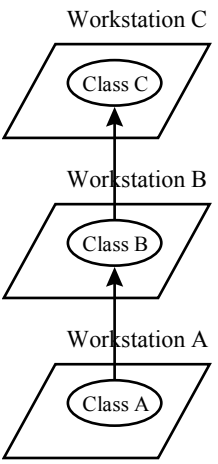


Fig. 1. *A typical example of a class hierarchy, where each class resides on a separate workstation within a distributed system*

If a message call is received by class A, the method that is contained in the message call could exist in either class B or class C which reside on different machines.

As dynamic binding requires that method lookup should take place at run-time, the time to locate a particular method in a distributed system would depend on the time it took to search local classes and the time it took to pass control and move data from workstation to workstation. In the case of the Smalltalk study [2], the fact that Smalltalk classes are dependent on a system hierarchy means that whenever a

Smalltalk class is moved to another workstation, details about the system hierarchy in which it resides must also be transferred. This adds to the amount of data being transferred.

The impetus for our re-evaluation of distributed class inheritance is the improvements in the support for distribution and the features of newer object-oriented languages which provide more suitable inheritance mechanisms for distributed systems. As the previous study of distributed class inheritance was based on Smalltalk, we argue that this study is not now representative, because Smalltalk is unique amongst object-oriented languages in the way that inheritance is used within the language. For example, any new class must be added to an existing system class hierarchy, where the hierarchy is part of the Smalltalk environment. It is therefore more difficult to distribute Smalltalk classes, as consideration must be given to the surrounding hierarchy in which the class resides. Furthermore, Smalltalk supports an inflexible form of single inheritance. New classes must inherit existing Smalltalk classes in order to derive system behaviour which provides support for such tasks as the creation of new objects. This explicit requirement to inherit system behaviour in this way, means that the inheritance feature in Smalltalk is confined to existing local classes. The inheritance of non-local or distributed classes therefore would not be a natural feature of such an inheritance mechanism. Newer object-oriented languages inherit system behaviour implicitly, thus allowing new classes to use the inheritance clause in a more flexible way, which is not dependent on local classes.

1.1. Distributed Class Inheritance Using Class Specifications

Our approach to distributed class inheritance is made possible through the use of class specifications. A class specification is made up of the interface of a class and the state declaration of a class. These elements represent the most portable parts of a class, because the interface and state declarations are mainly type representations, where types can usually be translated into machine independent representations. Machine independent type representations such as the eXternal Data Representation (XDR) from Sun [4] and CORBA marshalling types [3] are example implementations which provide type conversion procedures for a range of data types.

We regard the implementation detail of the behaviour of a class to be less portable than the interface and state declaration because the implementation is more machine dependent. This is due to many reasons, such as ownership, the configuration of supporting system software and more importantly architectural dependence, where a piece of source code is compiled or translated into object code, the object code is a software representation of the underlying architecture of that particular machine.

Although Java is regarded as a machine independent language, the lack of a standard implementation means that different machines could be using different versions of Java, so class implementations between different machines could still differ.

Distributed class inheritance using *class specification inheritance*, differs from other distributed inheritance models such as *object inheritance* [10] and *interface inheritance* [9], because it provides a more tightly coupled link between inherited classes. In a single-user system, classes which have been inherited by another, can be regarded as tightly-coupled because each of the inherited classes are used to construct

a single autonomous class which offers unique behaviour. Coupling of classes differs from static or dynamic binding, as binding deals with mechanisms for finding and locating the whereabouts of classes.

The semantic differences that result when class A inherits class B using these three distributed inheritance models can be seen in fig. 2. To highlight these differences, class A and class B are represented in terms of their public interface, their state representation and the implementation of their behaviour.

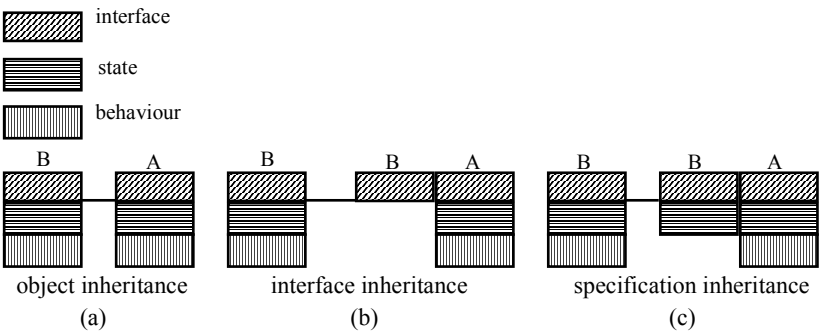


Fig. 2. *Shows diagrammatic models of three common models that are used to implement distributed inheritance.*

In part (a) of fig. 2, although class A and class B are disjoint, a semantic link exists between both classes, where class A is able to delegate [20] message calls to class B.

In part (b) of fig. 2, this represents interface inheritance where the interface of class B is physically attached to class A. The interface provides class A with a formal description of the behaviour of class B. As the interface of class B is tightly-coupled with the interface of class A, an external view of this class gives the impression that the behaviour of class A has been extended with that of class B. The semantic link that is derived from this form of inheritance however, is mainly syntactic because it enables classes like A, to perform local type-checking to ensure type conformance. As the implementation of class A and class B are loosely-coupled, this constrains the amount of control that is provided by interface inheritance. For example, the behaviour of class B cannot be customised by class A to enable class A to adapt inherited behaviour. Also, with interface inheritance, one cannot selectively inherit behaviour from an inherited class.

In part © of fig. 2, this shows specification inheritance, where the interface and the state representation of class B is physically attached to class A. There are a number of advantages that are gained by coupling the state representations in this way. One advantage is that class A will assume ownership of the inherited class. This means it can control and use the state representation for its own purpose. This differs from the other inheritance models, where the inherited state can be altered by a number of different objects who inherit the same class. With specification inheritance, one can also selectively inherit or modify the behaviour of a superclass. As the behaviour of a class is mainly used to affect the state of a class, because the state is localised, it enables support for alterations to be made to the inherited behaviour.

Distributed class inheritance using class specifications therefore promises a higher level of reuse than interface or object inheritance. We shall examine the use of this model of distributed class inheritance in more detail in subsequent sections, in order to determine the cost in performance of providing this additional level of reuse.

2. Experiment Test-Bed

The aim of this experiment was to assess the performance degradation that results from using distributed class inheritance. This assessment was achieved by developing a small DOIS, which included specification inheritance. Using specification inheritance, this would enable remote information based objects to be inherited by local information based objects. To examine the overheads of enabling this, message calls would be sent to the inherited information based object to determine the additional time that was needed in order to handle the call. Additional time would be incurred because even though the remote information base object would appear to reside on the client, its implementation would reside on the server Various timings were recorded in order to determine the level of degradation that resulted when different remote information based objects were inherited by a client.

The experiment was based around a simple dictionary application, where a client class called ComputerDictionary inherits a server class called Dictionary, both client and server were on separate machines. Although the ComputerDictionary and Dictionary classes were on separate machines, stub and skeleton classes were used to hide the underlying code which enabled support for distribution.

Fig. 3, provides a diagrammatic view of the relationship and location of the main classes that are used in this application.

In our implementation of distributed class inheritance, the server classes such as the dictionary class would continue to reside on the server. To establish a relationship between a client subclass and a server superclass, each client would receive a stub representation which would contain the specification of the server superclass. When a message call was sent to the Dictionary stub, the message call together with the values which make up the state would be packaged along with any arguments. The resultant data block would be sent to the server and be handled by the skeleton class. The skeleton class would unpack the state data and arguments and make a local call to the required method within the server superclass. When the call returns, the skeleton class would pack the state data along with any results and then send these back to the client.

This approach allows different clients to inherit the same server class, where the behaviour of the server class would be shared amongst each of the clients. As each client has their own unique copy of the state representation of the inherited server class, each client would assume ownership of that class.

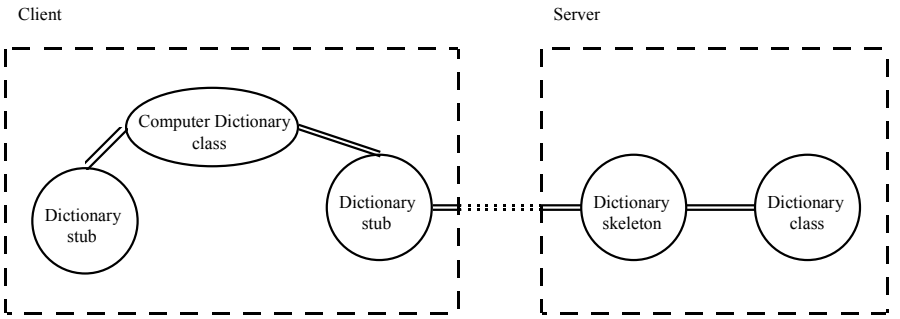


Fig. 3. *Captures the main client and server classes that are used in our model of inheritance.*

Using this simple application we can measure the performance degradation that is caused as a result of applying and using specification inheritance.

2.1 Equipment

The experiments were performed on an UltraSparc-1, model 170, running SUNOS 5.1 with a 167 MHz Ultra Sparc processor.

Two object-oriented languages were used in this study, namely, Eiffel [13] and Java [7]. The simple dictionary application was written separately in both these languages with the aim of determining the impact on the level of degradation that resulted from using a particular language, and whether one language was more efficient than the other. The Eiffel compiler that was used is called SmallEiffel version 0.98, which is a free Eiffel compiler that is available for Unix systems. The Java interpreter is based on JDK 1.1.2 which is also a free version that is available for Unix systems.

The network techniques that were used in conjunction with these languages were the remote method invocation (RMI) [8] method, CORBA [12] and the remote procedure call (RPC) [16] as they provide support to enable object inheritance, interface inheritance and specification inheritance to be used. The RMI implementation is available with the JDK 1.1.2 interpreter, the RPC implementation is bundled with the SUNOS operating system. The CORBA implementation that was used is OrbixWeb 3.0 [1] which is available as a free evaluation copy. These techniques were used in conjunction with the chosen languages, where the combinations which resulted, became known as EiffelRPC for using SmallEiffel with RPC; JavaRPC for using Java with RPC; JavaCorba for using Java with OrbixWeb and JavaRMI for using Java with RMI. It was not possible to use Eiffel with RMI and CORBA since RMI is specific to Java and the implementation of CORBA that was used was also based around Java.

2.2 Test Strategy

Various timings were taken to obtain details on: the overall time to process one message call; the length of time it took to marshal and unmarshal data; the length of time that the call spent travelling along the network; the length of time it took for a client to bind to a server; the length of time it took for the server to execute the call.

The timings were obtained by using the C library function called ftime. This provides a reliable and consistent value, where the time that was returned by ftime is in milliseconds. To use this function call in each of the languages, customised methods were written which made an external call to the same C function. Both Java and SmallEiffel provided in-built support for enabling calls to an external 'C' function.

To provide the same transport mechanism, each of the network paradigms used TCP/IP with SOCK_STREAM as the protocol, and the sender and receiver socket queue sizes were set to the maximum of 64K bytes. The data buffer sizes were kept to their default size, where the incoming data was read into a buffer which could handle up to 1024 bytes, and the outgoing data could accommodate up to 2048 bytes. These buffer sizes represented the most optimal settings for transporting data in this analysis, because the data would vary in size from a small number of bytes to a large number of bytes.

Timings were taken at distinct and identical stages in each of the experiments that was performed. This was to measure the impact of using different data sizes in each of the two languages using the two different network techniques namely, RMI and RPC. In all, about thirteen stages were identified, fig. 4 and fig. 5 captures each of these stages, where fig. 4 shows the stages on the client and fig. 5 shows the stages on the server.

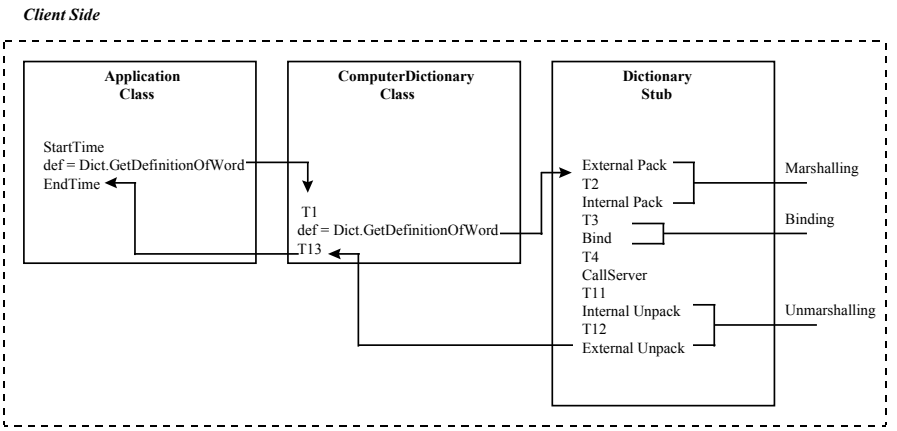


Fig. 4. *Highlights the various timing stages that were used to record timings on the client side.*

To ensure consistency, the GetMeaningOfWord method was used in each of the experiments, this method would be inherited by the client subclass, but its implementation would remain on the server. The call to this method would originate

at the client end and undergo a number of refinements before it is passed to the server. In the dictionary stub, the external pack describes the section where the state data of the inherited superclass is packaged into a single structure. This section is labelled external, because the functions which perform this task were distinct from the internal packaging functions that are used by each of the network techniques. The internal pack is specific to each of the network techniques, in the case of RPC, XDR is used, in the case of CORBA, the CORBA conversion methods are used and similarly in RMI, which has its own type conversion methods. When the call returns from the server, a reverse process takes place with the incoming data.

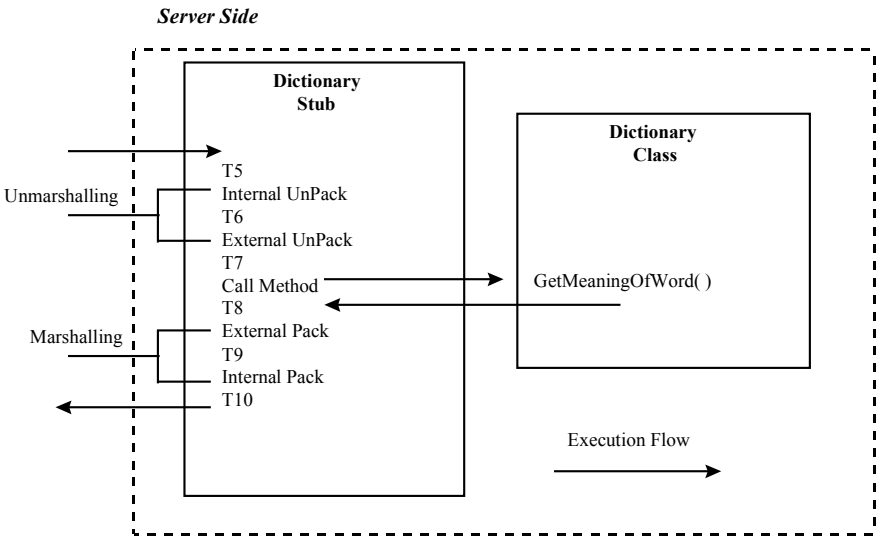


Fig. 5. *Highlights the various timing stages that were used to record timings at the server side.*

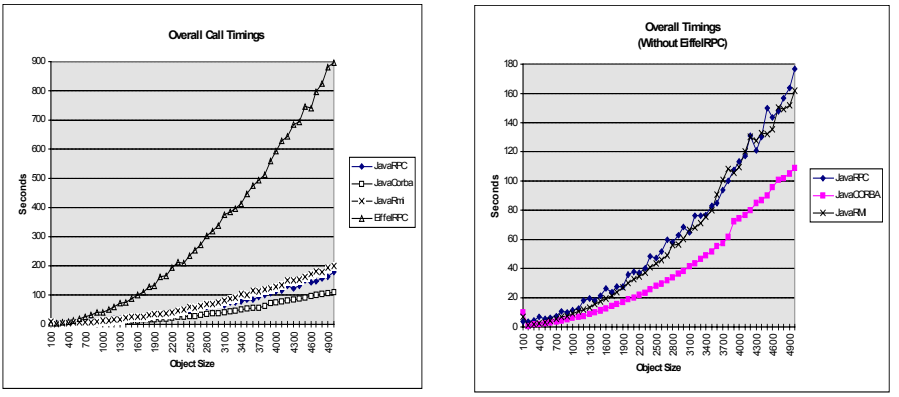
At the server side, much of the work that has to be performed is similar to the work stages that are performed within the stub class. Internal and external unpacking ensure that the copied state data and any arguments are translated into a similar type structure that is supported on the server. Such type conversions are used to re-build the client call on the server, to convert the remote message call to a local message call.

As the client code and the server code are based on different machines, it meant that two individual clocks had to be used to provide one measure. To handle the time differences between these clocks, any operation which spanned both machines had to catered for. For example, the length of time that the data spent in the network, was derived from subtracting the time spent on the client and the time spent on the server from the overall time. The overall time was determined by subtracting the completion time from the start time. The start and completion time are marked in figure 2 as StartTime and EndTime respectively. This gave a good estimate as the times were taken immediately prior to the call and immediately after the call. To ensure that the network was unaffected by any other processing the tests were performed in isolation, i.e. when no other users were logged into the network. To ensure this state, other users were prevented from using the network while the tests were underway.

3. Performance Results

The performance results are presented in a line graph format. The times which make up the graph were obtained from recording the length of time it took for a message call to be sent and processed by the server superclass. To ensure the timing value was a representative value, the same message call was sent ten times, the average of these samples was then used as the recorded value. To assess the performance degradation, the size of the state representation of a superclass which is copied to a client, was altered. In the dictionary class for example, the state could hold one or more words and their definitions. To make the state representation grow in this application, more words and definitions were added. Overall, the state representation grew by a factor of 10, where recordings were taken of objects ranging from 100 bytes in size up to 5000 bytes in size.

In fig. 6, the graphs represent the overall times that a message call that originated on the client took to complete. The same message call was sent along with different sized objects in order to analyse the effects. The overall timings provide an indication of the degree of degradation that occurs when objects of increasing size are passed over the network. One would expect some degradation to occur due to the increasing work that is needed, such as the marshalling and communication overheads. In part (a) of fig. 6, this graph shows the overall time in seconds, where the same application has been implemented in Java and Eiffel using the three different network techniques, RMI, CORBA and RPC. The differences are quite dramatic, where for example, in JavaCorba it takes 110 seconds to send a 5,000 byte object compared to using EiffelRPC, which takes 900 seconds to send the same object. After analysis, these differences can be attributed to the language that is being used and not to the network technique. Consider the performance of EiffelRPC and JavaRPC in graph (a) of figure 4, although both implementations use the same network technique, the overall timings are considerable different. The differences between each of the three network techniques can be seen in graph (b), of fig. 6, where Java has been used to implement each of these mechanisms. The use of JavaCorba provides the best performance of the three Java implementations. Further analysis of these figures taken for JavaCorba, suggest that the improved performance is due to the efficiency and seamlessness of Corba in translating object data into a representation that can be transported along a network. As the CORBA implementation is based on Java, the marshalling process is simply a selection mechanism, where appropriate conversion methods are used to place the data into an input or outputstream. In JavaRMI, the marshalling process uses object serialisation, where each object is packaged in a distinct format, where data, methods and arguments are copied into an input and outputstream. JavaRMI also uses dynamic class loading whenever the client stub or server skeleton is required. Prior to loading these classes, a search is performed to determine exactly which stub or skeleton class should be used. Dynamic class loading therefore, places an additional overhead compared to the overheads in CORBA.



(a) (b)

Fig. 6. *Captures graphically, the overall timings for a call which originates on the client, it is then passed to the server side for processing and the results are returned back to the client.*

In fig. 7, graphs (a) and (b) represent the amount of time that it takes for a message call to travel along the network. Each entry in the graph represents the length of time it took for the client to send data to the server together with the length of time that it took for the data to be sent from a server to a client. The graphs in fig. 7, highlight the differences between each of the network protocols and the transport mechanisms that exist within each of the networking techniques. In graph (a), clearly CORBA demonstrates the worst performance, where it takes 14 seconds for a 5,000 byte object to be sent and returned along a network. This poor performance can be attributed to the interoperable Internet InterOrb Protocol (IIOP) [4], which places additional overheads on a call. For example, a sequence of handshaking calls are sent to a server prior to committing a data packet to that particular server. The handshaking is a feature of IIOP and it is used to ensure that a particular server contains the implementation of a server object. Although these additional overheads provide a reliable and secure way of sending data along a network, the overheads incur heavy performance costs, as can be seen clearly in graph (a) of fig. 7.

Graph (b) of fig. 7, provides a more detailed view of the other approaches, where the imbalance caused by CORBA has been removed. With the Java and Eiffel implementations of RPC, there is very little difference between them. The RMI implementation however, is much slower in comparison. This is due to the setting up of connections to establish a reliable link and the overheads incurred by serialisation have impacted on the length of time it takes to transport data across a network. The fact that RPC provides the most efficient means of transporting data means that the greater level of security or reliability that is attributed to a network technique, the more the performance of that network technique is affected.

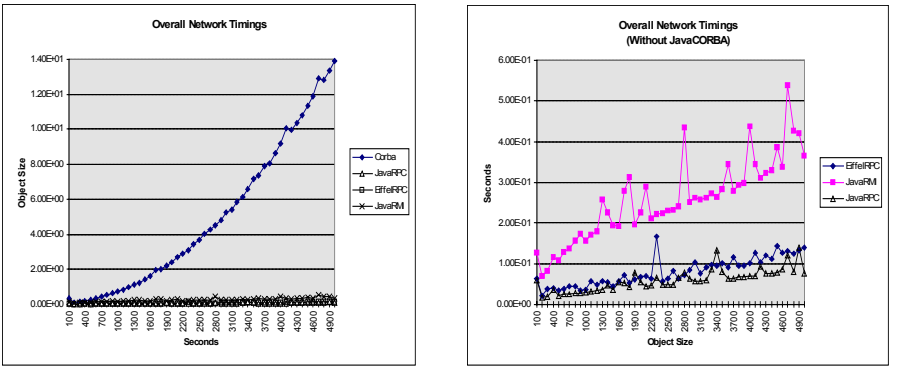


Fig. 7. *Shows graphically, the network timings which represent the length of time that a call and any associated data travels on the network, between a client and a server.*

In fig. 8, graphs (a) and (b) represent the length of time that it takes for a client to bind to a host. These times are for static binding only, where in each of the binding mechanisms within each of the network techniques, the name of the host is known. In the implementation of CORBA, one also had to specify a time which the Orbix daemon uses for the purposes of time-out. If no activity has taken place with a client that had previously bound to the host after the time-out value, then the host would break the connection with the client. To re-establish a connection with the host, the client would have to re-bind. In our application, the time-out value in the Orbix daemon was kept to the default value of 3,000 milliseconds.

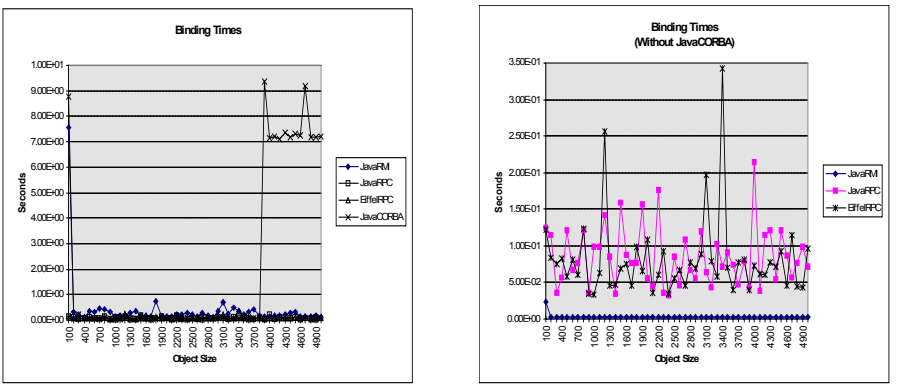
In graph (a) of fig. 8, by far the worst performance is represented by CORBA, where initially it takes 9 seconds for the client to bind to the host. The binding process in CORBA involves the conversion of a string into an interoperable object reference (IOR) which is then mapped to an object implementation. To accomplish this, the Orbix daemon when it receives a call, searches an interface repository to obtain a match with the string name that is contained in the bind operation. Once a match is found, the daemon then searches the implementation repository for the implementation of the named object. Although the binding process in CORBA can be optimised through the use of persistent objects, the creation of persistent objects would have an increasing affect in the use of non-persistent objects. Persistent objects would assume that any incoming object call is directed to them, only after this diversion would the call be sent to non-persistent objects.

The time-out feature in CORBA is highlighted in graph (a), where at the point where the object is 3,900 bytes in size, the client has to re-bind because the Orbix daemon has disconnected. As the client takes more than 3,000 milliseconds to marshal/unmarshal an object of 3,900 bytes in size, the daemon assumes that the client has broken off contact, the daemon therefore applies the time-out clause. One could increase the time-out value in an attempt to avoid this, but as object sizes vary, there is no way of knowing what optimum value to set as the time-out value.

In graph (b) of fig. 8, the imbalance caused by CORBA is removed, where the graph shows the binding times for each of the other approaches. Overall, the JavaRMI approach provides the most consistent time for binding, where the minimum time is 0.03 seconds and the maximum time is 0.12 seconds. In RMI, each host runs a

registry program which is used to monitor calls on a selected port. Whenever a client tries to bind to a host, the registry would inspect the call and try to match up the object name with the object name that had been previously registered. As the name lookup in RMI does not involve searching of external files as in CORBA, the lookup process is much faster.

In RPC, the RPC service is registered with a portmapper, where a unique program number, unique version number and a unique routine number is used to identify the RPC service. During a bind, the portmapper is used to locate the required RPC service. The binding process therefore requires little work and the level of the service depends on the availability of the portmapper. From the timings in graph (b) of fig. 8, there is not much difference between RMI and RPC. There are fluctuations with RPC, but these can be attributed to the delays with the portmapper. Unlike the RMI registry, the portmapper is a general purpose operating system feature which is used by system calls as well as calls originating from applications.



(a) (b)

Fig. 8. *Shows graphically the time it takes for a client to bind to a server using the three network techniques.*

What is evident from these timings is that the overheads of binding are small compared to the packing times which are shown in fig. 8. Furthermore, each mechanism provides facilities to optimise binding in order to tailor the performance for a particular application. Such optimisation would reduce the overheads accordingly.

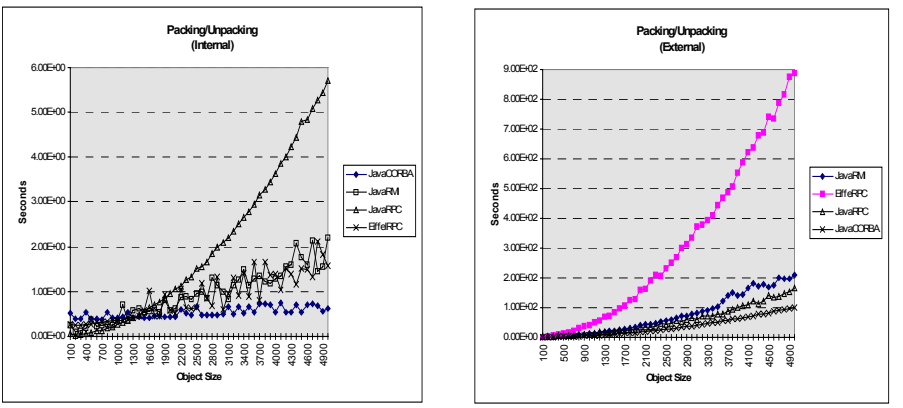
Graph (a) of fig. 9, represents the total length of time that each of the network techniques take when marshalling/unmarshalling data ready for transmission along the network. The fact that JavaRPC shows the most inefficient time, results from the fact that Java and RPC are based on different type structures. Java is based on the unicode character encoding [21], which is based on a 16-bit representation. To use Java with RPC, one has to convert unicode to the 8-bit ASCII representation that is used by the C language.

It is surprising in graph (a), that JavaCORBA is seen as a more efficient approach than JavaRMI. As RMI is regarded as a feature of the Java language one would expect the integration between the language and the network protocol to be closer

than JavaCORBA, as Java and CORBA are not as embedded as Java and RMI. This difference amounts to only 1 second between the amount of time it takes to marshalling/unmarshalling a 5000 byte object, so it is not a large difference.

In Graph (b) of fig. 9, this shows the length of time that it takes for a language to package the state data of an object into a single structure which can then be passed to the marshalling/unmarshalling functions of the network technique. The variations of the Java implementations are consistent, as the timings which are shown in graph (b) is quite similar in each. The slight variations which occur as the object grows is a result of the way Java dynamically loads classes from backing store. The common classes which are shared by these implementations, such as the clock class, reside at different directories than the implementations. The time that it takes to load these shared classes would depend on the locality of these classes and hence leads to the fluctuations in time.

The poor performance of SmallEiffel which was mentioned earlier is highlighted in graph (b), where there is a difference of upto 8 seconds between SmallEiffel and Java when processing 5000 bytes of data. This highlights the ineffciency of the language when it has to handle large quantities of data.



(a) *(b)*

Fig. 9. *Provides a graphical representation of the amount of time it takes to pack and unpack data on both the client and server.*

4. Analysis of Recordings

We analyse the level of impact that distribution had on this application, by considering the timings that were recorded for the binding, packing/unpacking and network timings which make up the overheads caused by distribution. Assuming that we can could optimise each network technique, so that we can derive the lowest times that is recorded by these tests. For example, the use of RPC provides the lowest

timings for network timings, CORBA provides the lowest timings for packing/unpacking and RMI provides the lowest times for binding.

Using these optimal timings, we can estimate the rate of growth, where objects of increasing size place an increasing burden on binding, packing/unpacking and network traffic. The rate of growth is estimated by line fitting each of the various graphs and finding the differential of each graph.

With the estimate for the rate of growth on network timings, the optimal timings are derived from the RPC network technique. This graph is shown in fig.10 together with the line that provides the best fit.

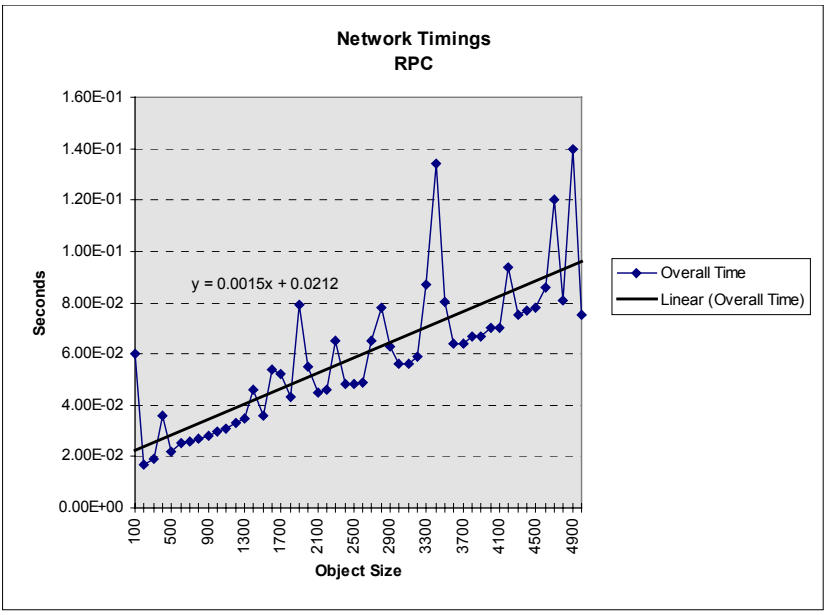


Fig. 10. *Shows a graphical representation of a linear best fit for the data that represents the network timings.*

The line in fig. 10 provides a rate of growth of 0.0015 seconds, so for each additional 100 bytes that is passed along the network it would take on average an extra 0.0015 seconds. This rate of growth is quite small and it would take up to a 100,000 byte object to reach a one second value.

Fig. 11, shows the graph representing the times for the internal packing/unpacking of data using CORBA marshalling.

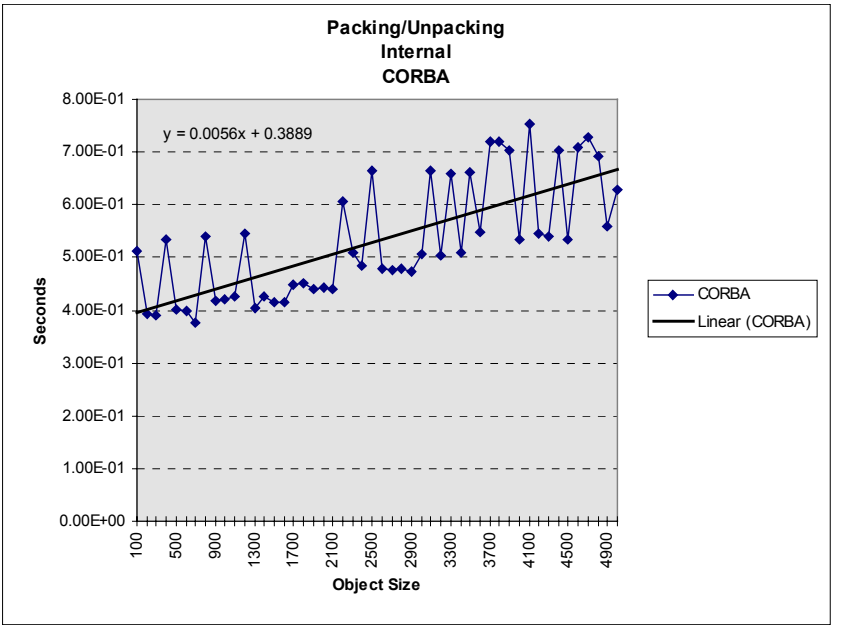


Fig. 11. *Shows a best fit linear line for the data that represents the packing and unpacking of data.*

From the line that best fits this graph, an estimated rate of growth is 0.0056 seconds for packing/unpacking each object of 100 bytes in size. This represents a very low overhead, where extremely large objects of 100,000 bytes or more would provide a figure over one second.

For the optimal times that represent binding, the rate of growth is not related to the size of the objects that are passed between client and server. From fig. 8, the graphs which represent the times for binding are fairly consistent, there is hardly any change when an object is 100 bytes compared to an object which is 5000 bytes.

5. Conclusions

By providing support for distributed class inheritance using inheritance by specification, this enables clients to use a single model of inheritance for both localised and distributed classes. By using a single model of inheritance, it allows both localised and distributed classes to be treated the same. This leads to a greater level of integration between local and distributed classes, where both types of class are able to inter-mix their behaviour. Localised classes are able to specialise the behaviour of inherited distributed classes so that the behaviour conforms to the existing behaviour of the local class.

As the specification of any inherited distributed class is copied to a client as a proxy, this enables support for substitutability and dynamic binding. The use of proxy

classes in place of the distributed classes, enable clients to build a local class hierarchy out of local and distributed classes. By enabling the construction of such a class hierarchy, this provides the necessary support for substitutability and dynamic binding.

The use of CORBA, RMI and RPC as implementation models for distributed class inheritance provided varying degrees of abstraction, where CORBA provides the highest level and RPC provides the lowest. Our tests show that there is a cost associated with the level of abstraction, where the higher level of abstraction the greater the cost, in terms of performance. One would expect some additional cost to enable support for heterogeneity, which is what is so distinctive about CORBA. Our findings show that this additional cost in performance is not unacceptable, where the overall overhead for supporting distributed class inheritance using these implementations is quite low. The most surprising outcome from the tests was that the choice of the language had a significant impact on the overall performance. This impact related mainly to how efficiently a language could process large amounts of data. This was apparent in the use of Java and Eiffel, where the version of Eiffel that we used proved to be inefficient in this area.

In the detailed analysis of CORBA, RMI and RPC, we found that each mechanism was efficient in different areas. The use of RMI and RPC for example, were more efficient at passing data along the network, whereas CORBA was more efficient at marshalling data prior to the sending and receiving of data from a network. To optimise the performance of such a model, one would anticipate that one could select the most efficient parts of one mechanism and then integrate these with other mechanisms. This requires that implementation models become more open to allow access to the individual parts of such a mechanism. Some researchers have already identified the need to make implementation models of distribution more open [14], thus allowing developers to choose the most efficient distribution mechanism at the time and enable distributed software systems to evolve in line with new developments. Examples of such developments is the idea that one can implement RMI on top of IIOP [6], thus enabling developers to integrate fully both Java and CORBA technologies. One further example is the use of objects by value instead of objects by reference, which is the focus of work that is being included in the CORBA 3 [18] standard.

To summarise, the analysis of the timings that were derived from using distributed inheritance show that the estimated rates of growth were minimal, and the impact on the performance of the application would only be affected if objects over 100,000 bytes in size were used. Such a small increase in degradation demonstrates that class inheritance could become an acceptable feature of distributed object systems. Our figures show that the poor performance is mainly attributed to an object-oriented language and not to the effects of distribution. With the continuous development of new and improved distribution mechanisms, one would expect this acceptability of inheritance in a distributed object system to be more attractive.

References

1. Baker S., "CORBA Distributed Objects Using Orbix", Addison-Wesley, ISBN 0-201-92475-7
2. Bennett J., "Distributed Smalltalk: Inheritance and Reactiveness in Distributed Systems", PhD Thesis, Department of Computer Science, University of Washington, December 1987
3. Betz, M.,"Networking Objects with CORBA", Dr Dobbs Journal, November 1995
4. Clip P., "IIOP: The Next HTTP?", BYTE Magazine, January 1998
5. Costa J., Sernadas A., Sernadas C.,"Object Inheritance beyond Subtyping", Acta Informatica, Vol. 31, No.1, January 1994
6. Curtis D., "White Paper on Java RMI and CORBA", Object Management Group, January 1997
7. Flannagan G., "Java in a Nutshell", 2nd Edition, O'Reilly Associates, 1996, ISBN: 1-56592-262-X
8. Hughes, M., et al., "Java Network Programming", Prentice-Hall, 1997, ISBN 0-13-841206-5
9. Leavens G.,"Inheritance of Interface Specification", ACM Sigplan Notices, vol. 29, no.8, August 1994
10. Malenfant J., "On the Semantic Diversity of Delegation based Programming languages", Proceedings of OOPSLA'95, Austin USA, 1995
11. Matsuoka S., Yonezawa A., "Analysis of Inheritance Anomaly in Object-Oriented Concurrent Programming Languages", Research Directions in COOP, Chapter 1, pages 107-150. MIT Press 1993
12. Otte R., Patrick P., Roy M., "Understanding CORBA, The Common Object Request Broker Architecture", Prentice-hall, 1996, ISBN:0-13-459884-9
13. Rine D., "Eiffel: A CASE tool supporting object-oriented software", ACM Sigsoft, Software Engineering Notes, vol. 17, no.1, January 1992
14. Snyder A., "Open Systems for Software: An Object-Oriented Solution", ACM OOPS Messenger, Addendum to the Proceddings of OOPSLA 1993, pp67-68, April 1994
15. Sun Microsystems Inc., "External Data Representation Protocol Specification", Sun Microsystems, 1986
16. Tay B., Arianda A., "A Survey of Remote Procedure calls", ACM Sigplan Notices, October 1992
17. Venners B., "Inheritance versus Composition: Which should you Choose", JavaWorld, Vol. 3, No. 1, 1998
18. Vinoski S., "New Features of CORBA 3.0", Communications of the ACM, Vol. 41, No. 10, October 1998
19. Wegner, P.,"Concepts and Paradigms of Object-oriented programming", Proceedings of OOPSLA'90 conference, ACM SIGPLAN Notices, June 1990
20. Wolczko, A.,"Encapsulation, Delegation and Inheritance in object-oriented languages", Software Engineering Journal, March 1992
21. Yourst M., "Inside Java Class Files", Dr. Dobb's Journal", January 1998

Performance Assessment Framework for Distributed Object Architectures

Matja B. Jurič, Tatjana Welzer, Ivan Rozman,
Marjan Heričko, Boštjan Brumen, Toma Domajnko, Aleš ivkovič

University of Maribor, Faculty of Electrical Engineering, Computer and Information Science, Institute of Informatics, Smetanova 17, SI-2000 Maribor
matjaz.juric@uni-mb.si

Abstract. Accurate, efficient and predictable performance assessment of distributed object models is necessary to make a founded decision about which model to use in a given application domain. This article presents a performance assessment framework for distributed object models. It presents two contributions to the study of distributed object performances: it defines the performance criteria for all important aspects of distributed object computing, including single and multi-client scenarios, and, it presents the high and low-level design of the framework and gives insights into implementation details for several important distributed object models, like CORBA, RMI and RMI-IIOP.

1. Introduction

Exponential network growth, global connectivity and new application domains have introduced important changes into the application development. Contemporary information systems are distributed and heterogeneous. Distributed systems require communication between computing entities. The communication can be established using low level mechanisms like sockets. For the new generation information systems high level approaches are needed.

Procedural languages have introduced a high level approach known as a remote procedure call (RPC). Pairing the RPC concept with the object paradigm results in a distributed object model. Today several distributed object models exist. The most important open models are Common Object Request Broker Architecture (CORBA) and Remote Method Invocation (RMI). Their basic functionality is similar - hiding the details of remote method invocations.

Choosing the appropriate distributed object model is a multi-criteria decision problem. Performance is a key criterion in the design, procurement, and use of computer systems [1]. This paper is focused on performance assessment of distributed object architectures. It defines the important performance criteria for distributed objects and presents a framework for gathering the quantitative data on which a comparative assessment of different distributed object models is possible. The paper describes the design and the implementation of the framework.

The presented framework represents many years of work and authors' experiences. It has outgrown the prototype state and is implemented for several distributed object

Eder et al. (Eds.): ADBIS´99, LNCS 1691, pp.349 -366, 1999.
© Springer-Verlag Berlin Heidelberg 1999

models: Inprise Visibroker, Iona Orbix, Expersoft PowerBroker, Java IDL (CORBA), RMI and RMI-IIOP, to name just a few. The practical value of the framework has been verified through the use by IBM Java Technology Centre, Hursley, England on the joint project with Sun: the development of RMI-IIOP, which will be included in the next version of JDK. The performance results obtained by using this framework have enabled identification of the most inefficient code and have helped to achieve up to six fold performance improvements.
The paper is organized into eight sections: Section 2 presents a short overview of the method invocation in distributed object models, Section 3 presents the related work and the goals, Section 4 gives a detailed overview of performance criteria, Section 5 and 6 present the design specifications, Section 7 the implementation insights and Section 8 gives the concluding remarks.

2. Method Invocation in Distributed Object Models

Distributed object models allow clients to invoke methods on distributed objects without concern for the following [2]:

- *Object location transparency:* the details about location of client and server objects are managed without affecting the implementation.
- *Platform and programming language transparency:* the client does not have to know on which platform runs the server nor which programming language it is implemented in.
- *Communications protocols, interconnections and hardware transparency:* distributed object models provide an abstraction layer. Therefore the application does not have to cope with different communication protocols, data type sizes, ranges, storage layouts and other hardware differences.

The client invokes a method on the remote server (object implementation) with a simple command, i.e., `object.method(args)`. The servers returns the result as a return value or through arguments. Several types of method invocations are possible, for example static and dynamic invocation, one- and two-way, synchronous, deferred synchronous and asynchronous invocation.
The integral part of a distributed object model is the object request broker (ORB) [2, 3, 4]. ORB is responsible for handling the details of the communication:

- marshalling the client's request,
- locating the appropriate target object,
- transferring the request,
- demultiplexing the request to the object adapter and the servant,
- dispatching the request and the parameters to the appropriate method (operation) of the server, and returning the result.

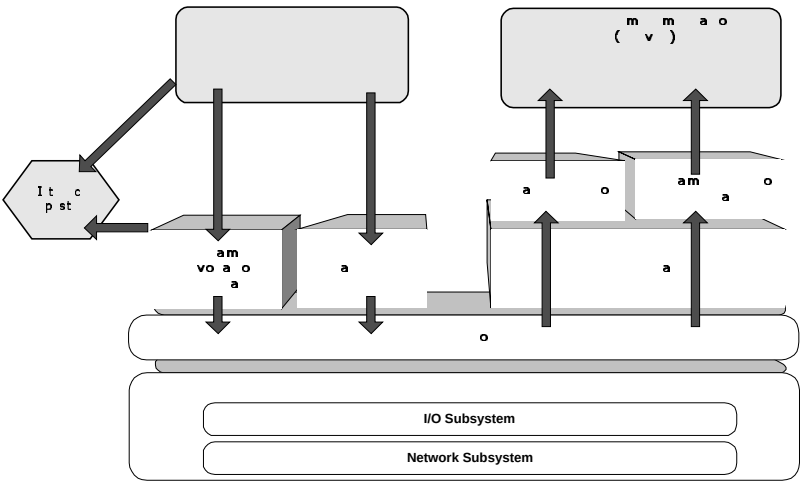


Fig. 1. Typical architecture of an object request broker

Figure 1 presents the typical architecture of an ORB. Client is a distributed object that performs application tasks by obtaining object references to remote server objects and invoking methods on them. Server (object implementation) is an instance of a class, associated with a public remote interface, that implements the services provided by the object. The process in which the object implementation is executed is called servant. The client/server roles can be (and often are) changed during the run-time. ORB core is responsible for delivering the request to the object and returning the response. The request can be initiated statically, when the client has compile time knowledge about the services provided by the server. It can also be initiated dynamically when the client builds the request in run-time. Then it uses the interface repository services. The server can be implemented using the static skeleton or a dynamic skeleton and can route the incoming request to the appropriate methods. To provide the communication, the ORB core uses the operating system services. Distributed method invocation is very complex and introduces many layers of overhead.

3. Objectives

3.1. Related Work

Evaluating the performance of distributed object models is a difficult task. The review of related work has shown that a performance assessment model for distributed object architecture does not exist. The research on performances is limited mostly to the CORBA architecture and C++ programming language. The majority of the work is focused on latency and scalability investigations, mostly over high-speed networks, where single client and single server configurations are used. In [5] the authors report the performance results from benchmarking sockets and several CORBA implementations over Ethernet and ATM networks. The paper also describes ACE. In

[6] the authors compared the performance of socked-based communication, RPC, Orbix and ORBeline over ATM network and discovered the sources of overhead. They used a single client and a single server configuration. In [7] the authors measured and explained the overhead of CORBA Dynamic Invocation Interface and Dynamic Skeleton Interface. In [8] and [9] the authors systematically analyzed the latency and scalability of Visibroker and Orbix and revealed the sources of overhead. Again they used a single client and server configuration over ATM network. They also described techniques to improve the performances and they gave an overview of TAO. In [10] the author described the implementation of a low overhead ORB. He presented some performance results where he used single client server configuration and C++ language. Some performance results in the context of real-time systems are presented in [11], [12] and [13]. The only performance investigations for multi-client scenarios, CORBA, RMI and Java have been published in [14] and [15]. A common characteristic of all performance studies is that the authors used very simple tests to measure performance and that they only investigated single client scenarios. The most advanced testing method has been used in [14] where multi-client scenarios were based on a prototype ATM (Automatic Teller Machine) application. The performance assessment framework described in this paper takes a systematic approach to performance evaluation and covers the majority of important criteria. It is designed to offer portability between different distributed object models and comparability of the results.

3.2. Goals

The performance assessment framework should identify a set of performance criteria that enable a quantitative performance evaluation of different object request broker (ORB) implementations. The criteria encompass the aspects of ORB performance that are important from the application developer's point of view. The criteria help in the selection and ensure the suitability of ORB middleware for the target application domain. Different application domains have different requirements with respect to performance criteria. Therefore the criteria should be carefully selected to satisfy the broad audience. They should be independent of an ORB implementation or architecture. They should cover important features like synchronous one- and two-way communication, deferred synchronous communication, static and dynamic method invocation, different threading models, exception handling, etc.

The performance assessment framework should provide design specifications for implementing the performance assessment applications. It should give clear instructions about the high and low level design, about kinds of interfaces, methods, data types and data sizes used. It should also give an exact specification of dynamic behavior which includes the performance measurement procedure and synchronization issues. The performance assessment framework should define the data patterns used for performance assessment and specify how to gather the performance results.

4. Performance Criteria

The performance assessment framework defines the following criteria which are important for quantitative evaluation of ORB performance:

- *Round trip time (RTT)* - measures the time needed from the point when the client initiates a method invocation to the point when the client receives the result. The processing on the server side is minimized.
- *Throughput* - measures the number of method invocations in a given time interval. Throughput and round trip time are inversely proportioned.
- *Data Throughput* - measures the ORB efficiency of data transfer. The data is transferred as parameters of the methods (which are sent from a client to the server) and as results from method invocations (which are transferred from the server to the client).
- *Scalability* - evaluates the performance degradation when multiple clients simultaneously interact with a single server object.
- *Multi-threading strategy* - evaluates the ability of the ORB to provide simultaneous method invocation on a single server object. ORB can support one or more threading models such as thread-per-servant, thread-per-connection, thread-per-request and a thread pool. The chosen model can have significant influence on performances.
- *Demultiplexing efficiency* - evaluates the efficiency of the object adapter to direct the incoming request to the appropriate servant. The demultiplexing procedure generally has two steps. The first step is from the object adapter to the target object and skeleton, and the second step is from the skeleton to the implementation method.
- *Dispatching overhead* - measures the overhead of the object adapter to locate the appropriate server.
- *Binding overhead* - evaluates the time overhead for obtaining the references to the remote objects using the services provided by the ORB.
- *Exception handling latency* - evaluates the cost of handling user defined exceptions.
- *Efficiency of the Interface Repository* - measures the interface repository querying performance.
- *Efficiency of the Dynamic Invocation Interface* - measures the efficiency of the dynamic method invocations compared to the static invocation mechanism.
- *Efficiency of the Dynamic Skeleton Interface* - measures the efficiency of the dynamic server implementations compared to the static skeleton implementations.

5. Design of Object Implementations

5.1. Round Trip Time, Throughput, Data Throughput

The performance assessment framework introduces several server side interfaces that enable gathering of information required to quantitatively represent the defined

performance criteria. For measuring the round trip time, throughput and data throughput interfaces with associated classes have been defined as shown in Figure 2.

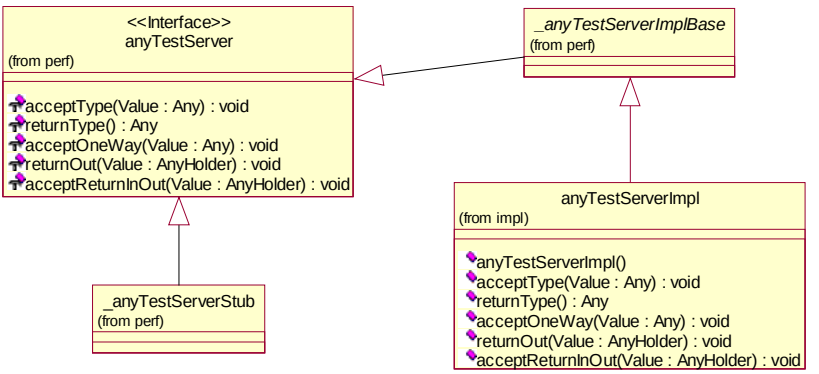


Fig. 2. The <data-type>TestServer interface and the implementation class

Each interface defines five methods:

- `acceptType` method accepts an argument of the measured data type and has no return,
- `returnType` method accepts no arguments and returns the data type,
- `acceptOneWay` method is similar to `acceptType` except that it uses the one way invocation mechanism,
- `returnOut` method is similar to `returnType` except that it returns the result through the *out* argument which is of selected data type,
- `acceptReturnInOut` method accepts an argument of the selected data type and returns the result through the same argument (*inout* type).

CORBA IDL data type	RMI data type
boolean	boolean
char	char
wchar	char
octet	byte
short	short
long	int
long long	long
float	float
double	double
string	String
wstring	String
any	Object

Table 1. Simple data types used in performance assessment framework

Interfaces and complying implementation classes are defined for the simple data types shown in Table 1. All the basic data types are covered. Also a special *any* data type is used, which can hold all the other simple data types.

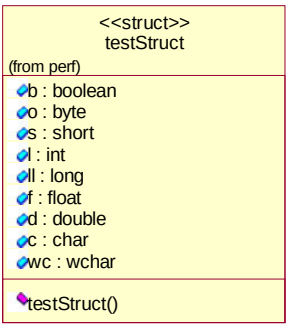


Fig. 3. The testStruct definition

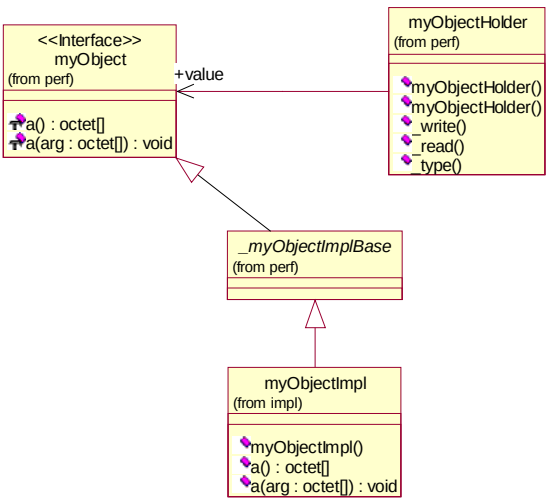


Fig. 4. The definition of myObject

In addition to simple data types two user defined data types are introduced. First, the structure `testStruct`, which is presented in Figure 3.
Second, a user defined class is introduced, named `myObject.` The definition is shown in Figure 4. Please notice the Holder class which enables the implementation of `returnOut` and `acceptReturnInOut` methods.
To be able to measure the data throughput several new interfaces are introduced. Again five methods are introduced. These methods accept and return sequences or arrays respectively. They do not deal with single instances of appropriate data types. The sequences for CORBA IDL and arrays for RMI are defined as follows:

```
typedef sequence<(data_type)> (data_type)Seq;
          // IDL
(data_type)[] ValueSeq;
          // RMI
```

Again for each simple data type, for the `testStruct` and `myObject` the template interfaces (shown in Figure 5) are defined.

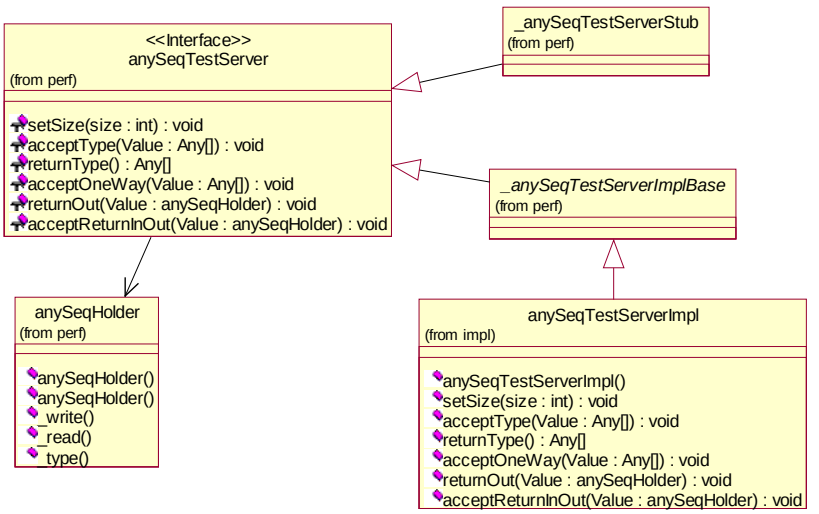


Fig. 5. The <data-type>SeqTestServer interface and the implementation class

The set of methods is similar to the already described interfaces, except that the return parameters are sequences and arrays, respectively. The added method `setSize` takes care of setting the right data size used for testing.

5.2. Scalability

Scalability is defined as performance degradation when multiple clients interact with a single server object. Therefore multiple clients should invoke the described methods simultaneously and without delays. Multiple clients can be executed each on its own computer device or each in its own process on a single computer device. It is necessary to assure synchronization between the clients. Therefore the performance assessment framework defines two interfaces for synchronization, shown in Figure 6. The interface `synchr` supervises the synchronization of multiple clients at the beginning of the tests and manages the number of simultaneous clients. To make the data gathering process easier the `synchr` interface implements the following functionality for the client number management:

- it takes care of the synchronization before the tests can begin,
- it takes care that all the clients wait for the last client to finish the tests, and
- it shuts down the last client and repeats the tests for *n-1* clients.

The interface `testSynchr` supervises the synchronization of different method invocations during the testing. The synchronization is achieved using semaphores and is done in the intervals between performance measurements. The implementation classes of both interfaces use the concurrency services to prevent the conflict situations caused when multiple clients simultaneously access a shared resource.

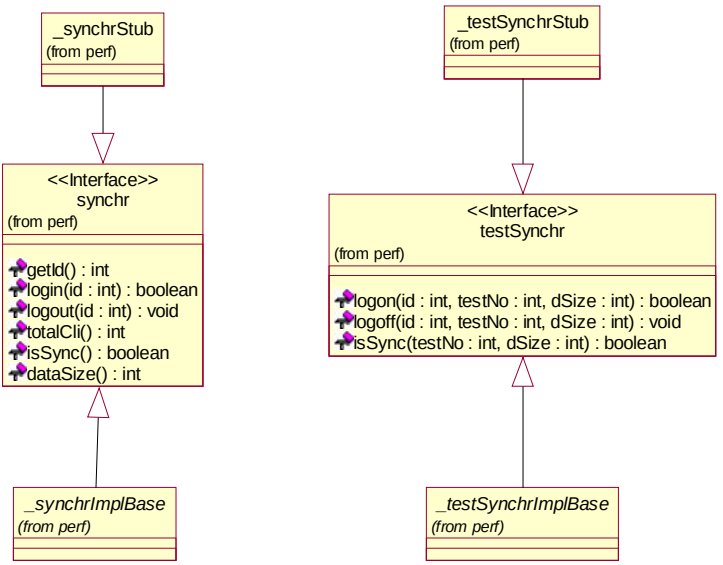


Fig. 6. Synchronization interfaces

To minimize the influence of synchronization on the performance results the synchronization interfaces are implemented in a separate servant that should be executed on a separate computer. The synchronization is implemented so that it minimizes the network traffic. Therefore and because of the fact that when synchronizing objects no time is measured the impact of the synchronization to the performance results can be neglected.

5.3. Multi-Threading Strategy

The testing method foresees that multiple clients invoke methods on a single server object. The multithreading strategy that is supported by the server object and the ORB is crucial and has an important impact on the performances. To determine the multithreading strategy used in the scenarios we have performed a simple test. We have defined a server interface that had one method (Figure 7). This method delays the program execution for 30 seconds.

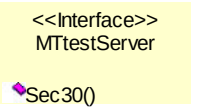


Fig. 7. Interface used for multi-threading evaluation

The method should be invoked: (1) by all simulations single-threaded clients, (2) by a single client that is multi-threaded. The execution time should be measured. The threading model should be selected as follows:

- Thread-per-servant: if the execution time of (1) is `no_clients * 30` seconds;
- Thread-per-connection: if the execution time of (1) is 30 seconds and the execution time of (2) is `no_clients * 30` seconds;
- Thread-per-request: if the execution time of (1) and (2) is 30 seconds;
- Thread pool: if the execution time of (1) and (2) is 30 seconds and if the requests in (1) and (2) get serialized after a specific number of requests.

5.4. Demultiplexing Efficiency

The demultiplexing efficiency is evaluated in two steps. First, an interface with 1024 methods has been defined. The names of the methods have the following format: *mNNNN*, where NNNN is a successive number. The subordination between the method invocation round trim time and the method number is investigated. Second, an interface with 1024 methods has been defined where the names differ in length. The subordination between the method invocation RTT and the length of method name is investigated. The interfaces are shown in Figure 8.

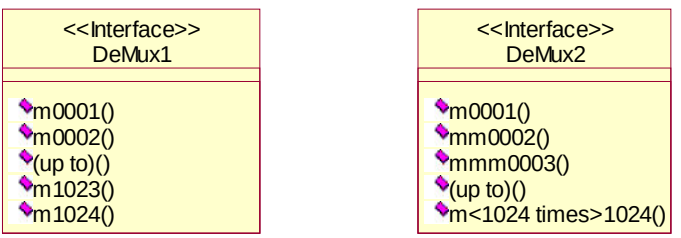


Fig. 8. Interfaces for demultiplexing efficiency evaluation

5.5. Dispatching Overhead

To measure the dispatching overhead multiple instances of the same server class are instantiated. The client invokes the methods on all the objects:

- sequentially from the first to the last object,
- sequentially form the last to the first object and
- in random order.

The dependency between the method invocation (RTT) time and the number of simultaneous server side object implementations is investigated. The difference in RTT times points to the caching algorithm used by the ORB.

5.6. Binding Overhead

Before the client object can use the services provided by the server object implementation it has to obtain the reference to the remote object. In CORBA compliant ORBs the following code is used:

```
NameComponent path[]={null};
org.omg.CORBA.Object objRef =
orb.resolve_initial_references("NameService");
NamingContext ncRef =
NamingContextHelper.narrow(objRef);
NameComponent nc = new NameComponent("myObject", "");
path[0] = nc;
myRef = myObjectHelper.narrow(ncRef.resolve(path));
```

In RMI the naming service is simpler and requires the URL address of the host computer:

```
myRef =
(myRefClass)Naming.lookup("//"+myHost+"/myObject");
```

Several CORBA implementations provide an alternative non-compliant way for getting the initial object reference. Presented example shows the non-compliant bind method generated in Visibroker's Helper classes:

```
myRef = myObjectHelper.bind(orb,"myObject");
```

5.7. Exception Handling Latency

The <data-type>TestServerImpl classes which implement <data-type>TestServer interfaces are modified so that the methods return a user defined exception rather than terminating normally. The difference between the client side RTT by the normal termination and by the exception scenario shows us the latency of returning an exception.

5.8. Efficiency of the Interface Repository and the Dynamic Invocation Interface

The client object is modified in a way that it does not have static knowledge about server side object implementations. Therefore it first queries the interface repository to get the information about the methods signatures. Then it dynamically constructs a request and initiates it. The client side RTT is compared to the static invocation mechanism used in previous tests. An example code for CORBA architecture is given:

```
// Interface Repository
org.omg.CORBA.ORB orb =
org.omg.CORBA.ORB.init(args,null);
org.omg.CORBA.Repository rep =
org.omg.CORBA.RepositoryHelper.bind(orb);
org.omg.CORBA.InterfaceDef idef =

org.omg.CORBA.InterfaceDefHelper.narrow(rep.lookup(idlN
ame));
org.omg.CORBA.InterfaceDefPackage.FullInterfaceDescript
ion intDesc =
          idef.describe_interface();

// Dynamic Invocation Interface
org.omg.CORBA.Request request =
osn._request("acceptType");
request.add_in_arg().insert_long((int)10);
request.set_return_type(orb.get_primitive_tc(org.omg.CO
RBA.TCKind.tk_void));
request.invoke();
```

5.9. Efficiency of the Dynamic Skeleton Interface

The server side implementation is modified. A dynamic scheduling method takes care of handling the `ServerRequest` object and invoking the appropriate method. This criterion is valid for CORBA compliant ORBs only. The difference in RTT for dynamic and static skeleton interface shows the overhead of the dynamic skeleton interface.

6. The Client Side Design

The actual measurements are done on the client side. A client implements the following activities:

- binds to the server side object implementations used for synchronization,
- binds to the server side object implementations used for performance measurements,
- opens an output table where the results are written in,
- allocates memory for storing the temporary results,
- performs the performance measurements,
- calculates the results,
- writes the results to the table.

6.1. Performing the Performance Measurements

The performance measurements are performed for each interface, described in the previous section. Let X denote the performance measure of interest. To obtain the *i*-th observation X_i, $i=1, 2, \ldots, n$, the following steps are necessary:

- the client logs on to the testSyncrh interface before each sequence of method invocations,
- the client waits to get the synchronization acknowledgement,
- it reads the system time,
- it performs the method invocation. To get the desired resolution of the result the test is performed *r* times,
- it reads the system time again and calculates the X_i-th performance observation.

After completing the *n* observations, sample mean $\underline{X}$, the variance s^2 and the standard deviation *s* are calculated.

6.2. Multi-Client Scenario

To enable multi-client scenarios the client side functionality has to be extended. Therefore the client should:

- logon to the `synchr` interface before performing each set of tests for a given number of simultaneous clients,
- logoff after the set of test is done and shutdown if the `synchr` instructs to do so.

Executing performance measurements with multiple clients can be done automatically. The user has to start the maximum number of clients. When the tests are done, the last client automatically shuts down and the test are repeated for *n-1* clients. This continues until the last client shuts down.

6.3. Determining the Necessary Number of Observations

The number of observations *n* required to achieve the results with an accuracy of $\pm r\%$ and a confidence level of $100(1 - \alpha)\%$ can be determined as follows. For a sample of size *n*, the $100(1 - \alpha)\%$ confidence interval of the population mean is (1)

(**1**)

$$\overline{\quad} \quad \sqrt{\quad}$$

where *s* is the standard deviation and *z* is the $(1 - \alpha/2)$-quantile of a unit normal variate of the desired confidence level [1].

The desired accuracy *r* implies that the confidence interval should be $(\underline{X}(1 - r/100), \underline{X}(1 + r/100))$. We can determine *n* using the following equations (2), (3):

$$\overline{\quad} \quad \sqrt{\quad} = \quad \frac{r}{\quad} \tag{2}$$

$$n = \left(\frac{s}{\underline{X}} \right)^2 \tag{3}$$

7. Implementation

The component diagram in Figure 9 shows the three components used for performance measurements. These three parts are the server component where all the object implementation used for performing the performance measurements are collected. The controller component takes care of synchronization, for setting the appropriate data sizes and other book keeping activities. It is important to understand that the synchronization component should be executed on a separate computer in order to minimize the influence on the performance measurement results. The client performs the actual method invocations, computes and stores the results. In the multi-client scenarios there are multiple instances of the client component. In a usual environment each client component is executed on a separate computer. Because of the already described characteristics of a distributed method invocation the deployment of the components on computers can vary without the need to change the code.

The performance assessment framework has been implemented for CORBA, RMI and RMI-IIOP. The following CORBA implementations were used: Inprise Visibroker, Iona Orbix, Expersoft PowerBroker and Java IDL (JDK 1.2). The framework has been first implemented in Java and then in C++ programming language. There is no hindrance to use other standardized (Smalltalk, Ada, COBOL, C, PLI) and non-standardized languages (Delphi, Objective C, Eiffel). During the implementation only the standardized features have been used. The vendor-specific extensions have been used only in the evaluation of the binding efficiency where beside the naming services the non-compliant `bind` method has been used. For the RMI the JDK 1.2 and JDK 1.1.7 from Sun have been used.

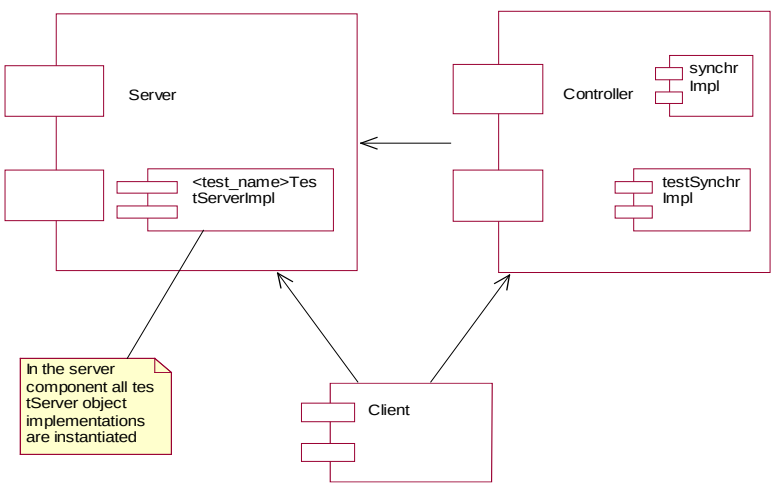


Fig. 9. Component diagram

RMI-IIOP is an extension for the Java platform that will enable the RMI to use the IIOP (Internet Inter-ORB Protocol) for inter-object communication. That will make

RMI protocol-compatible with CORBA and will enable arbitrary interoperability between RMI and CORBA objects. RMI has been developed by a joint project between IBM and Sun at IBM Java Technology Centre, England. Through our cooperation with this IBM center we were able to implement the framework for an early beta version of the RMI-IIOP. The syntax has not been fully defined therefore several modifications to the RMI have been necessary and we had to use the beta interfaces that will probably change in the release version. With the performance results the identification of the most time consuming parts of the coda has been possible. Several weaknesses have been removed, among other the very ineffective transmission of object references. The purpose of this paper prohibits us to describe the details of optimizations.

To get relevant performance results that are reproducible it is necessary to minimize the impact of the environment, this means the factors of hardware, operating system and network. It is necessary that no other tasks are running on the computers used for testing and that the network is free of other traffic. Results are comparable only if the same computing equipment has been used for all tests, if the operating system and the network are identical. To simulate the performance of a real-world application it is however possible to run the tests in a real-world environment.

The performance assessment framework can be adapted to specific needs. It can be used to predict the real-world performances of distributed applications in a given environment. Therefore ways should exist to calibrate the model. Only a well calibrated framework can predict the real world application performances. This framework can be fully calibrated in the following ways:

- the methods can be modified so that they match the target application domain,
- the data samples can be modified so that they match the actual data interaction between the objects,
- the interaction between distributed objects can be calibrated,
- the number of simultaneous clients can be matched to the target environment.

Figure 10 presents a screen shot of a running performance assessment application. The four windows are the server, the controller, the client component, and the name server.

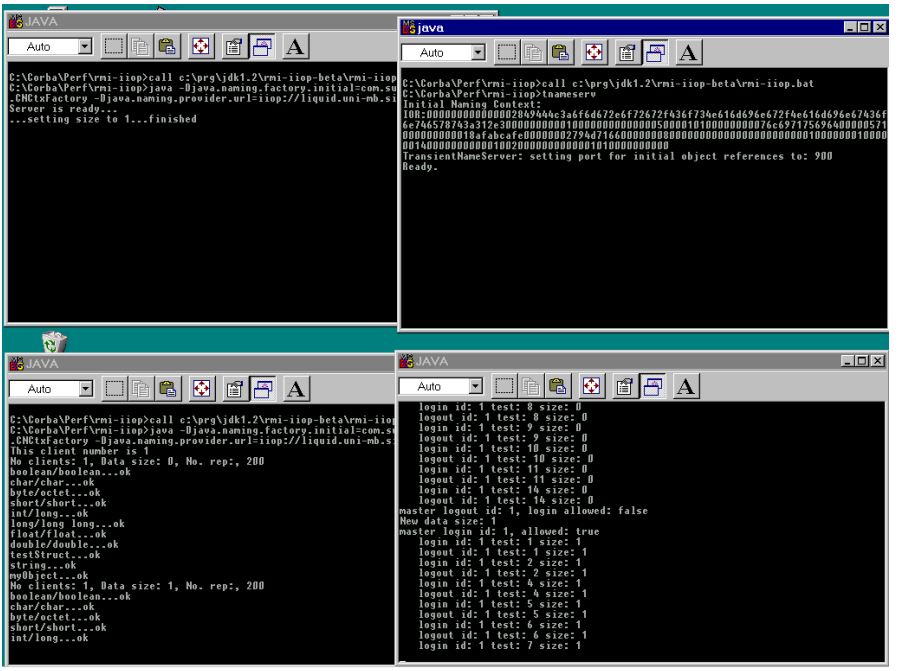


Fig. 10. A screen shot of a running performance assessment application.

The results are gathered in the format that is presented in Figure 11 is used. They can be stored into any database that supports the ODBC/JDBC interface.

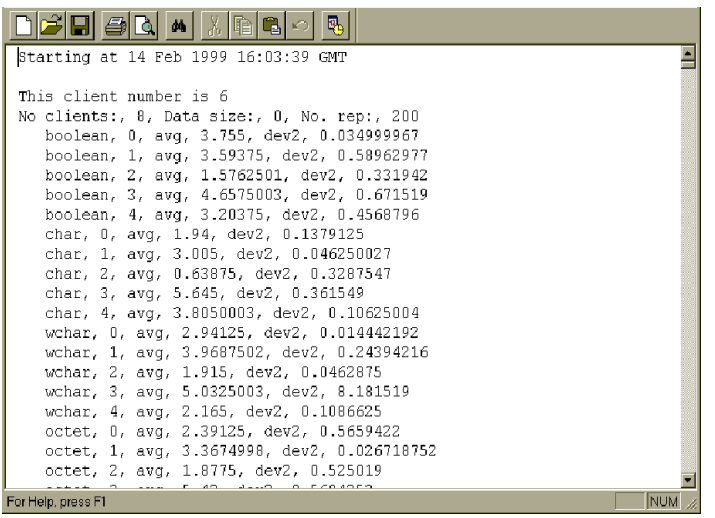


Fig. 11. Sample results

8. Conclusion

Performance is an important criterion in the design, procurement, and use of distributed object systems. Assessing the performance of distributed object models is a difficult task and very little has been done in this area. In the article the performance assessment framework for distributed object models has been presented. The framework has clearly defined the performance criteria in an implementation independent way. Therefore it enables the comparison of the performance results for different, architecturally diversified distributed object models. Further, detailed high and low-level design specification have been presented and implementation details for the most important open distributed object models have been shown. Performance assessment framework is implemented for the majority of important distributed object models and has proven its value by the development of RMI-IIOP.

The presented framework forms a good basis for further research. On one side we are working on supplementing the framework with formal decision model that will support the selection process. We will also extend the framework by a detailed specification for data analysis and calibration. On the other side the results gathered with this framework open possibilities for evaluating and comparing different distributed object models and identifying the inefficient parts of the systems. They also make it possible to validate analytical models.

References

[1] Kant K., Introduction to Computer System Performance Evaluation, McGraw-Hill Computer Sciences Series, 1992

[2] Object Management Group, The Common Object Request Broker: Architecture and Specification, Revision 2.2, February 1998

[3] Sun Microsystems, Inc., Java Remote Method Invocation Specification, Sun, February 1997

[4] Object Management Group, Richard Mark Soley, Christopher M. Stone, Object Management Architecture Guide, John Wiley & Sons, Inc., 1995

[5] Douglas C. Schmidt, Tim Harrison, Ehab Al-Shaer, Object-Oriented Components for High-speed Network Programming, 1st Conference on Object-Oriented Technologies, USENIX , Monterey, CA, June, 1995

[6] Aniruddha S. Gokhale, Douglas C. Schmidt, Measuring the Performance of Communication Middleware on High-Speed Networks, SIGCOMM Conference, ACM 1996, Stanford University, August 28-30, 1996

[7] Aniruddha S. Gokhale, Douglas C. Schmidt, The Performance of the CORBA Dynamic Invocation Interface and Dynamic Skeleton Interface over High-Speed ATM Networks, IEEE GLOBECOM '96 conference, November 18-22nd, 1996, London, England

[8] Aniruddha S. Gokhale, Douglas C. Schmidt, Evaluating CORBA Latency and Scalability Over High-Speed ATM Networks, IEEE 17th International Conference on Distributed Systems (ICDCS 97), May 27-30, 1997, Baltimore, USA

[9] Aniruddha S. Gokhale, Douglas C. Schmidt, Measuring and Optimizing CORBA Latency and Scalability Over High-Speed Networks, IEEE Transactions on Computers, Vol. 47, No. 4, April 1998

[10] Sai-Ali Lo, The Implementation of a Low Call Overhead IIOP-based Object Request Broker, Olivetti & Oracle Resharch Laboratory, April 1997 (www.orl.co.uk)
[11] Douglas C. Schmidt, Aniruddha Gokhale, Timothy H. Harrison, Guru Parulkar, A High-performance Endsystem Architecture for Real-time CORBA, Distributed Object Computing, IEEE Communications Magazine, Vol. 14, No. 2, February 1997
[12] Christopher D. Gill, David L. Levine, Douglas C. Schmidt, The Design and Performance of a Real-Time CORBA Scheduling Service, August 1998
[13] Ashish Singhai, Aamod Sane, Roy Campbell, Reflective ORBs: Supporting Robust, Time-critical Distribution, ECOOP'97 Workshop Proceedings, 1997
[14] Hericko Marjan, Juric B. Matjaz, Zivkovic Ales, Rozman Ivan, Java and Distributed Object Models: An Analysis, ACM SIGPLAN Notices, December 1998
[15] Juric B. Matjaz, Zivkovic Ales, Rozman Ivan, Are Distributed Objects Fast Enough, JavaREPORT, SIGS Publications, May 1998

Efficient Use of Signatures in Object-Oriented Database Systems

Kjetil Nørvåg

Department of Computer and Information Science
Norwegian University of Science and Technology
7491 Trondheim, Norway
noervaag@idi.ntnu.no

Abstract. Signatures are bit strings, generated by applying some hash function on some or all of the attributes of an object. The signatures of the objects can be stored separately from the objects themselves, and can later be used to filter out candidate objects during a perfect match query. In an object-oriented database system (OODB) using logical object identifiers (OIDs), an object identifier index (OIDX) is needed to map from logical OID to the physical location of the object. In this paper we show how signatures can be stored in the OIDX, and used to reduce the average object access cost in a system. We also extend this approach to transaction time temporal OODBs (TOODB), where this approach is even more beneficial, because maintaining signatures comes virtually for free. We develop a cost model that we use to analyze the performance of the proposed approaches, and this analysis shows that substantial gain can be achieved.

1 Introduction

A signature is a bit string, which is generated by applying some hash function on some or all of the attributes of an object.[1] When searching for objects that match a particular value, it is possible to decide from the signature of an object whether the object is a possible match. The size of the signatures is generally much smaller than the size of the objects themselves, and they are normally stored separately from the objects themselves, in signature files. By first checking the signatures when doing a perfect match query, the number of objects to actually be retrieved can be reduced.

Signature files have previously been shown to be an alternative to indexing, especially in the context of text retrieval [1,6]. Signature files can also be used in general query processing, although this is still an immature research area.

The main drawback of signature files, is that signature file maintenance can be relatively costly. If one of the attributes contributing to the signature in an object is modified, the signature file has to be updated as well. To be beneficial, a high read to write ratio is necessary. In addition, high selectivity is needed at query time to make it beneficial to read the signature file in addition to the objects themselves.

[1] Note that *signatures* are also often used in other contexts, e.g., function signatures and implementation signatures.

Eder et al. (Eds.): ADBIS'99, LNCS 1691, pp. 367–381, 1999.
© Springer-Verlag Berlin Heidelberg 1999

In this paper, we first show how signatures can be incorporated into traditional object-oriented databases (OODB). Second, we show how they can be used in *transaction time temporal OODBs*, with only marginal maintenance cost.

Every object in an OODB is uniquely identified by an object identifier (OID). To do the mapping from logical OID to physical location, an OID index (OIDX), often a B-tree variant, is used.[2] The entries in the OIDX, which we call *object descriptors* (OD), contains the physical address of the object. Because of the way OIDs are generated, OIDX accesses often have low locality, i.e., often only one OD in a particular OIDX leaf node is accessed at a time. This means that OIDX lookups can be costly, but they have to be done every time an object is to be accessed (as will be explained later, the lookup cost can be reduced by employing OD caching). OIDX updates are only needed when object are created, moved, or deleted. It is not necessary when objects are updated, because updates to object are done in-place, so that the mapping information in the OIDX is still valid after an object have been updated.

Our approach to reduce the average access cost in the system, is to include the signature of an object *in the OIDX itself*. This means that the OD now also includes the signature, in addition to the mapping information. When we later do a value based perfect match search on a set, we can in many cases avoid retrieving the objects themselves, checking the signature in the OD is enough to exclude an object during the search. The OD will have to be retrieved anyway, because it is needed to find the physical location of the object, so there is no additional cost to retrieve the OD, compared to not using signatures. Storing the signature in the OIDX increases the size of the OD, and the size of the OIDX, and makes an OIDX update necessary every time an object is updated, but as we will show later in this paper, in spite of this extra cost, it will in most cases be beneficial.

A context where storing signatures in the OIDX is even more interesting, is transaction time temporal OODBs (TOODB). In a TOODB, object updates do not make previous versions unaccessible. On the contrary, previous versions of objects can still be accessed and queried. A system maintained timestamp is associated with every object version. This timestamp is the commit time of the transaction that created this version of the object. In a non-temporal OODB, the OIDX update would not be necessary if we did not want to maintain signatures. In a TOODB, on the other hand, *the OIDX must be updated every time an object is updated*, because we add a new version, and the timestamp and the physical address of the new version need to be inserted into the index. As a result, introducing signatures only marginally increases the OIDX update costs. Because of the low locality on updates, disk seek time dominates, and the increased size of the ODs is of less importance. *With this approach, we can maintain signatures at a very low cost, and by using signatures, one of the potential bottlenecks in an TOODB, the frequent and costly OIDX updates, can be turned into an advantage!*

The organization of the rest of the paper is as follows. In Sect. 2 we give an overview of related work. In Sect. 3 we give a brief introduction to signatures. In Sect. 4 we describe indexing and object management in a TOODB. In Sect. 5 we describe how

[2] Some OODBs avoid the OIDX by using physical OIDs. In that case, the OID gives the physical disk page directly. While this potentially gives a higher performance, it is very inflexible, and makes tasks as reclustering and schema management more difficult.

signatures can be integrated into OID indexing. In Sect. 6 we develop a cost model, which we use in Sect. 7 to study the performance when using signatures stored in the OIDX, with different parameters and access patterns. Finally, in Sect. 8, we conclude the paper and outline issues for further research.

2 Related Work

Several studies have been done in using signatures as a text access methods, e.g. [1,6]. Less has been done in using signatures in ordinary query processing, but studies have been done on using signatures in queries on set-valued objects [7].

We do not know of any OODB where signatures have been integrated, but we plan to integrate the approaches described in this paper in the Vagabond parallel TOODB [9].

3 Signatures

In this section we describe signatures, how they can be used to improve query performance, how they are generated, and signature storage alternatives.

A signature is generated by applying some hash function on the object, or some of the attributes of the object. By applying this hash function, we get a signature of F bits, with m bits set to 1. If we denote the attributes of an object i as $A_1, A_2, \ldots, A_n$, the signature of the object is $s_i = S_h(A_j, \ldots A_k)$, where S_h is a hash value generating function, and $A_j, \ldots A_k$ are some or all of the attributes of the object (not necessarily including all of $A_j, \ldots A_k$). The size of the signature is usually much smaller than the object itself.

3.1 Using Signatures

A typical example of the use of signatures, is a query to find all objects in a set where the attributes match a certain number of values, $A_j = v_j, \ldots, A_k = v_k$. This can be done by calculating the query signature s_q of the query: $s_q = S_h(A_j = v_j, \ldots, A_k = v_k)$ The query signature s_q is then compared to all the signatures s_i in the signature file to find possible matching objects. A possible matching object, a *drop*, is an object that satisfies the condition that all bit positions set to 1 in the query signature, also are set to 1 in the object's signature. The drops forms a set of candidate objects. An object can have a matching signature even if it does not match the values searched for, so all candidate objects have to be retrieved and matched against the value set that is searched for. The candidate objects that do not match are called *false drops*.

3.2 Signature Generation

The methods used for generating the signature depend on the intended use of the signature. We will now discuss some relevant methods.

Whole Object Signature. In this case, we generate a hash value from the whole object. This value can later be used in a perfect match search that includes all attributes of the object.

One/Multi Attribute Signatures. The first method, *whole object signature*, is only useful for a limited set of queries. A more useful method is to create the hash value of only one attribute. This can be used for perfect match search on that specific attribute. Often, a search is on perfect match of a subset of the attributes. If such searches are expected to be frequent, it is possible to generate the signature from these attributes, again just looking at the subset of attributes as a sequence of bits. This method can be used as a filtering technique in more complex queries, where the results from this filtering can be applied to the rest of the query predicate.

Superimposed Coding Methods. The previous methods are not very flexible, they can only be used for queries on the set of attributes used to generate the signature. To be able to support several query types, that do perfect match on different sets of attributes, a technique called *superimposed coding* can be used. In this case, a separate attribute signature for each attribute is created. The object signature is created by performing a bitwise OR on each attribute signature, for an object with 3 attributes the signature is $s_i = S_h(A_0)$ OR $S_h(A_1)$ OR $S_h(A_2)$. This results in a signature that can be very flexible in its use, and support several types of queries, with different attributes.

Superimposed coding is also used for fast text access, one of the most common applications of signatures. In this case, the signature is used to avoid full text scanning of each document, for example in a search for certain words occurring in a particular document. There can be more than one signature for each document. The document is first divided into logical blocks, which are pieces of text that contain a constant number of distinct words. A word signature is created for each word in the block, and the block signature is created by OR-ing the word signatures.

3.3 Signature Storage

Traditionally, the signatures have been stored in one or more separate files, outside the indexes and objects themselves. The files contains s_i for all objects i in the relevant set. The sizes of these files are in general much smaller than the size of the relation/set of objects that the signatures are generated from, and a scan of the signature files is much less costly than a scan of the whole relation/set of objects. Two well-know storage structures for signatures are *Sequential Signature Files* (SSF) and *Bit-Sliced Signature Files* (BSSF). In the simplest signature file organization, SSF, the signatures are stored sequentially in a file. A separate *pointer file* is used to provide the mapping between signatures and objects. In an OODB, this file will typically be a file with OIDs, one for each signature. During each search for perfect match, the whole signature file has to be read. Updates can be done by updating only one entry in the file.

With BSSF, each bit of the signature is stored in a separate file. With a signature size F, the signatures are distributed over F files, instead of one file as in the SSF approach. This is especially useful if we have large signatures. In this case, we only have to search the files corresponding to the bit fields where the query signature has a "1". This can reduce the search time considerably. However, each update implies updating up to F files, which is very expensive. So, even if retrieval cost has been shown to be much smaller for BSSF, the update cost is much higher, 100-1000 times higher is not

uncommon [7]. Thus, BSSF based approaches are most appropriate for relatively static data.

4 Object and Index Management in TOODB

We start with a description of how OID indexing and version management can be done in a TOODB. This brief outline is not based on any existing system, but the design is close enough to make it possible to integrate into current OODBs if desired.

4.1 Temporal OID Indexing

In a traditional OODB, the OIDX is usually realized as a hash file or a B^+-tree, with ODs as entries, and using the OID as the key. In a TOODB, we have more than one version of some of the objects, and we need to be able to access current as well as old versions efficiently. Our approach to indexing is to have *one* index structure, containing all ODs, current as well as previous versions.

In this paper, we assume one OD for each object version, stored in a B^+-tree. We include the commit time *TIME* in the OD, and use the concatenation of OID and time, *OID*||*TIME*, as the index key. By doing this, ODs for a particular OID will be clustered together in the leaf nodes, sorted on commit time. As a result, search for the current version of a particular OID as well as retrieval of a particular time interval for an OID can be done efficiently. When a new object is *created*, i.e., a new OID allocated, its OD is appended to the index tree as is done in the case of the Monotonic B^+-tree [5]. This operation is very efficient. However, when an object is *updated*, the OD for the new version *has to be inserted into the tree.*

While several efficient multiversion access methods exist, e.g., TSB-tree [8], they are not suitable for our purpose, because they provide more flexibility than needed, e.g., combined key range and time range search, at an increased cost. We will never have search for a (consecutive) range of OIDs, OID search will always be for *perfect match*, and most of them are assumed to be to the current version. It should be noted that our OIDX is inefficient for many typical temporal queries. As a result, additional secondary indexes can be needed, of which TSB-tree is a good candidate. However, *the OIDX is still needed*, to support navigational queries, one of the main features of OODBs compared to relational database systems.

4.2 Temporal Object Management

In a TOODB, it is usually assumed that most accesses will be to the current versions of the objects in the database. To keep these accesses as efficient as possible, and to benefit from object clustering, the database is partitioned. Current objects reside in one partition, and the previous versions in the other partition, in the *historical database*. When an object is updated in a TOODB, the previous version is first moved to the historical database, before the new version is stored in-place in the current database.

We assume that clustering is not maintained for historical data, so that all objects going historical, i.e., being moved because they are replaced by a new version, can be

written sequentially, something which reduces update costs considerably. The OIDX is updated *every time an object is updated*.

Not all the data in a TOODB is temporal, for some of the objects, we are only interested in the current version. To improve efficiency, the system can be made aware of this. In this way, some of the data can be defined as non-temporal. Old versions of these are not kept, and objects can be updated in-place as in a traditional OODB, and the costly OIDX update is not needed when the object is modified. This is an important point: using an OODB which efficiently supports temporal data management, should not reduce the performance of applications that do not utilize these features.

5 Storing Signatures in the OIDX

The signature can be stored together with the mapping information (and timestamp in the case of TOODBs) in the OD in the OIDX. Perfect match queries can use the signatures to reduce the number of objects that have to be retrieved, only the candidate objects, with matching signatures, need to be retrieved.

Optimal signature size is very dependent of data and query types. In some cases, we can manage with a very small signature, in other cases, for example in the case of text documents, we want a much larger signature size. It is therefore desirable to be able to use different signature sizes for different kind of data, and as a result, we should provide different signature sizes.

The maintenance of object signatures implies computational overhead, and is not always required or desired. Whether to maintain signatures or not, can for example be decided on a per class basis.

6 Analytical Model

Due to space constraints, we can only present a brief overview of the cost model used. For a more detailed description, we refer to [10]. The purpose of this paper is to show the benefits of using signatures in the OIDX, so we will restrict this analysis to attribute matches, using the superimposed coding technique.

Our cost model focus on disk access costs, as this is the most significant cost factor. In our disk model, we distinguish between random and sequential accesses. With random access, the time to read or write a page is denoted T_P, with sequential access, the time to read or write a page is T_S. All our calculations are based on a page size of 8 KB.

The system modeled in this paper, is a page server OODB, with temporal extensions as described in the previous sections. To reduce disk I/O, the most recently used index and object pages are kept in a *page buffer* of size M_{pbuf}. OIDX pages will in general have low locality, and to increase the probability of finding a certain OD needed for a mapping from OID to physical address, the most recently used ODs are kept in a separate OD cache of size M_{ocache}, containing N_{ocache} ODs. The OODB has a total of M bytes available for buffering. Thus, when we talk about the memory size M, we only consider the part of main memory used for buffering, not main memory used for the

objects, the program itself, other programs, the operating system, etc. The main memory size M is the sum of the size of the page buffer and the OD cache, $M = M_{\text{pbuf}} + M_{\text{ocache}}$.

With increasing amounts of main memory available, buffer characteristics are very important. To reflect this, our cost model includes buffer performance as well, in order to calculate the hit rates of the OD cache and the page buffer. Our buffer model is an extension of the Bhide, Dan and Dias LRU buffer model [2]. An important feature of the BDD model, which makes it more powerful than some other models, is that it can be used with *non-uniform access distributions*. The derivation of the BDD model in [2] also includes an equation to calculate the number N_d of distinct objects out of a total of N access objects, given a particular access distribution. We denote this equation $N_{distinct}(N_d, N)$. The buffer hit probability of an object page is denoted P_{buf_opage}. Note that even if index and object pages share the same page buffer, the buffer hit probability is different for index and object pages. We do not consider the cost of log operations, because the logging is done to separate disks, and the cost is independent of the other costs.

To analyze the use of signatures in the OIDX, we need a cost model that includes:

1. OIDX update and lookup costs.
2. Object storage and retrieval costs.

The OIDX lookup and update costs can be calculated with our previously published cost model [10]. The only modification done to this cost model is that signatures are stored in the object descriptors (ODs). As a consequence, the OD size varies with different signature sizes. In practice, a signature in an OD will be stored as a number of bytes, and for this reason, we only consider signature sizes that are multiples of 8 bits.

The average OIDX lookup cost, i.e., the average time to retrieve the OD of an object, is denoted $T_{\text{oidx\_lookup}}$, and the average time to do an update is $T_{\text{oidx\_update}}$. Not all objects are temporal, and in our model, we denote the fraction of the operations done on temporal objects as P_{temporal}. For the non-temporal objects, if signatures are not maintained, the OIDX is only updated when the objects are created.

6.1 Object Storage and Retrieval Cost Model

One or more objects are stored on each disk page. To reduce the object retrieval cost, objects are often placed on disk pages in a way that makes it likely that more than one of the objects on a page that is read, will be needed in the near future. This is called clustering. In our model, we define the clustering factor C as the fraction of an object page that is relevant, i.e., if there are N_{o_page} objects on each page, and n of them will be used, $C = \frac{n}{N_{o\_page}}$. If $N_{o\_page} < 1.0$, i.e., the average object size is larger than one disk page, we define $C = 1.0$.

Read Objects. We model the database read accesses as 1) *ordinary object accesses*, assumed to benefit from the database clustering, and 2) *perfect match queries*, which can benefit from signatures. We assume the perfect match queries to be a fraction P_{qm} of the read accesses, and that P_A is the fraction of queried objects that are actual drops. Assuming a clustering factor of C, the average object retrieval cost, excluding OIDX

lookup, is $T'_{readobj} = \frac{1}{CN_{o\_page}} T_{readpage}$, where the average cost of reading one page from the database, is $T_{readpage} = (1 - P_{buf\_opage})T_P$. When reading object pages during signature based queries, we must assume we can not benefit from clustering, because we retrieve only a very small amount of the total number of objects. In that case, one page must be read for every object that is retrieved, $T''_{readobj} = T_{readpage}$. The average object retrieval cost, employing signatures, is:

$$T_{readobj} = T_{\text{oidx\_lookup}} + (1 - P_{qm})T'_{readobj} + P_{qm}(P_A T''_{readobj} + (1 - P_A)F_d T''_{readobj})$$

which means that of the P_{qm} that are queries for perfect match, we only need to read the object page in the case of actual or false drops. The false drop probability when a signature with F bits is generated from D attributes is denoted $F_d = (\frac{1}{2})^m$, where $m = \frac{F \ln 2}{D}$.

Update Objects. Updating can be done in-place, with write-ahead logging (but note that in the case of temporal objects, these are moved to the historical partition before the new current version is written). In that case, a transaction can commit after its log records have been written to disk. Modified pages are not written back immediately, this is done lazily in the background as a part of the buffer replacement and checkpointing. Thus, a page may be modified several times before it is written back.

Update costs will be dependent of the checkpoint interval. The checkpoint interval is defined to be the number of objects that can be written between two checkpoints. The number of written objects, N_{CP}, includes created as well as updated objects. $N_{CR} = P_{new} N_{CP}$ of the written objects are creations of new objects, and $(N_{CP} - N_{CR})$ of the written objects are updates of existing objects.

The number of distinct updated objects during one checkpoint period is $N_{DU} = N_{distinct}(N_{CP} - N_{CR}, N_{obj})$. The average number of times each object is updated is $N_U = \frac{N_{CP} - N_{CR}}{N_{DU}}$. During one checkpoint interval, the number of pages in the current partition of the database that is affected is $N_P = \frac{N_{DU}}{N_{o\_page} C}$ This means that during one checkpoint interval, new versions must be inserted into N_P pages. CN_{o_page} objects on each page have been updated, and each of them have been updated an average of N_U times. For each of these pages, we need to write $P_{\text{temporal}} N_U CN_{o\_page}$ objects to the historical partition (this includes objects from the page and objects that were not installed into the page before they went historical), install the new current version to the page, and write it back. This will be done in batch, to reduce disk arm movement, and benefits from sequential writing of the historical objects. For each of the object updates, the OIDX must be updated as well. In the case of a non-temporal OODB, we do not need to write previous versions to the historical partition, and the OIDX needs only to be updated if signatures are to be maintained.

When new objects are created, an index update is needed. When creating a new object, a new page will, on average, be allocated for every N_{o_page} object creation. When a new page is allocated, installation read is not needed. The average object update cost, excluding OIDX update cost:

$$T_{write\_obj} = T_{\text{oidx\_update}} + \frac{N_P T_S (P_{\text{temporal}} N_U CN_{o\_page}) + N_P T_P + \frac{N_{CR}}{N_{o\_page}} T_P}{N_{CP}}$$

Note that objects larger than one disk page will usually be partitioned, and each object is indexed by a separate large object index tree. This has the advantage that when a new version is created, only the modified parts need to be written back. An example of how this can be done is the EXODUS large objects [3].

7 Performance

We have now derived the cost functions necessary to calculate the average object storage and retrieval costs, with different system parameters and access patterns, and with and without the use of signatures. We will in this section study how different values of these parameters affect the access costs. Optimal parameter values are dependent of the mix of updates and lookups, and they should be studied together. If we denote the probability that an operation is a write, as P_{write}, the average access cost is the weighted cost of average object read and write operations:

$$T_{\text{access}} = (1 - P_{\text{write}})T_{\text{lookup}} + P_{\text{write}}T_{\text{update}}$$

Our goal in this study is to minimize T_{access}. We measure the gain from the optimization as: Gain $= 100\big(\frac{(T^{nonopt}_{\text{access}} - T^{opt}_{\text{access}})}{T^{opt}_{\text{access}}}\big)$ where $T^{nonopt}_{\text{access}}$ is the cost if not using signatures, and T^{opt}_{access} is the cost using signatures.

Access Model The access pattern affects storage and retrieval costs directly and indirectly, through the buffer hit probabilities. The access pattern is one of the parameters in our model, and is modeled through the independent reference model. Accesses to objects in the database system are assumed to be random, but skewed (some objects are more often accessed than others), and the objects in the database can be logically partitioned into a number of partitions, where the size and access probability of each partition can be different. With β_i denoting the relative size of a partition, and α_i denoting the percentage of accesses going to that partition, we can summarize the four access patterns used in this paper:

Set	β_0	β_1	β_2	α_0	α_1	α_2
3P1	0.01	0.19	0.80	0.64	0.16	0.20
3P2	0.001	0.049	0.95	0.80	0.19	0.01
2P8020	0.20	0.80	-	0.80	0.20	-
2P9505	0.05	0.95	-	0.95	0.05	-

In the first partitioning set, we have three partitions. It is an extensions of the 80/20 model, but with the 20% hot spot partition further divided into a 1% hot spot area and a 19% less hot area. The second partitioning set, 3P2 resembles the access pattern close to what we expect it to be in future TOODBs. The two other sets in this analysis have each two partitions, with hot spot areas of 5% and 20%.

Parameter	Value	Parameter	Value	Parameter	Value
M	100 MB	S_{obj}	128	C	0.3
M_{ocache}	0.2 M	N_{objver}	100 mill.	D	1
N_{CP}	$0.8N_{\text{ocache}}$	S_{od}	$32 + \lceil \frac{F}{8} \rceil$	P_{new}	0.2
P_A	0.001	P_{write}	0.2	P_{qm}	0.4
P_{temporal}	0.8				

Table 1. Default parameters.

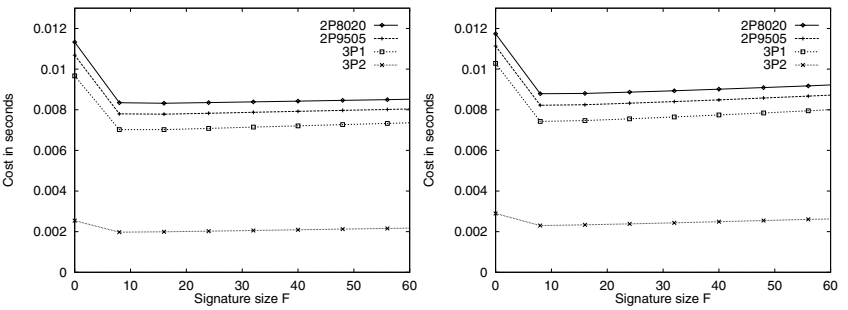


Fig. 1. Cost versus signature size for different access patterns. Non-temporal OODB to the left, temporal OODB to the right.

Parameters We consider a database in a stable condition, with a total of N_{objver} objects versions (and hence, N_{objver} ODs in the OIDX). Note that with the OIDX described in Section 4.1, OIDX performance is not dependent of the number of existing versions of an object, only the total number of versions in the database.

Unless otherwise noted, results and numbers in the next sections are based on calculations using default parameters, as summarized in Table 1, and access pattern according to partitioning set 3P1.

With the default parameters, the studied database has a size of 13 GB. The OIDX has a size of 3 GB in the case of a non-temporal OODB, and 5 GB in the case of a temporal OODB (not counting the extra storage needed to store the signatures). Note that the OIDX size is smaller in a non-temporal OODB, because in a non-temporal OODB, we do not have to store timestamps, and we have no inserts into the index tree, only append-only. In that case, we can get a very good space utilization [4]. When we have inserts into the OIDX, as in the case of a temporal OODB, we get a space utilization in the OIDX that is less than 0.67.

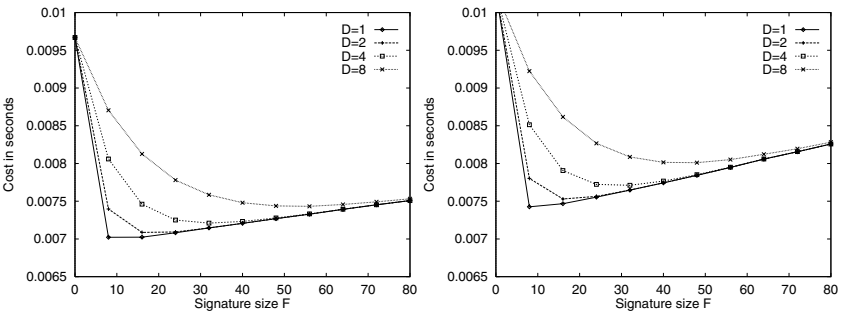


Fig. 2. Cost versus signature size for different values of D. Non-temporal OODB to the left, temporal OODB to the right.

7.1 Optimal Signature Size

Choosing the signature size is a tradeoff. A larger signature can reduce the read costs, but will also increase the OIDX size and OIDX access costs. Figure 1 illustrates this for different access patterns. In this case, a value of $F = 8$ seems to be optimal. This is quite small, and gives a higher false drop probability than accepted in text retrieval applications. The reason why such a small signature is optimal in our context, is that the size of objects is small enough to make object retrieval and match less costly than a document (large object) retrieval and subsequent search for matching word(s), as is the case in text retrieval applications.

The signature size is dependent of D, the number of attributes contributing to the signature. This is illustrated in Fig. 2. With an increasing value of D, the optimal signature size increases as well. In our context, a value of D larger than one, means that more than one attribute contributes to the signature, so that queries on more than one attribute can be performed later.

In the rest of this study, we will use F=8 when using the default parameters, and use F=16, 32 and 48 for D=2, 4, and 8, respectively.

7.2 Gain from Using Signatures

Figure 3 shows the gain from using signatures, with different access patterns. Using signatures is beneficial for all access patterns, except when only a very limited amount of memory is available.

Figure 4 shows the gain from using signatures, for different values of D. The gain decreases with increasing value of D.

7.3 The Effect of the Average Object Size

We have chosen 128 as the default average object size. It might be objected that this value is too large, but Fig. 5 shows that even with smaller object sizes, using signatures will be beneficial. The figure also shows how the gain increases with increasing object size.

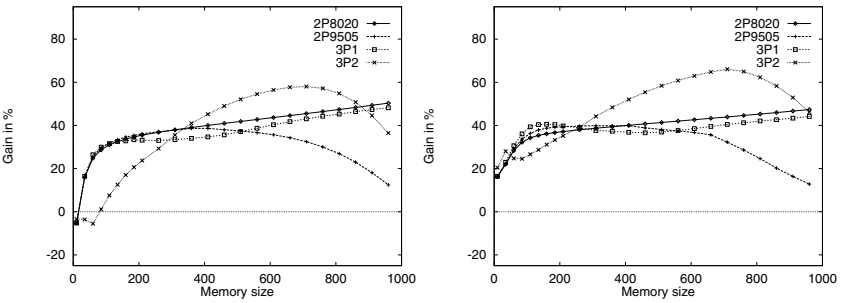


Fig. 3. Gain from using signatures versus memory size, for different access patterns. Non-temporal OODB to the left, temporal OODB to the right.

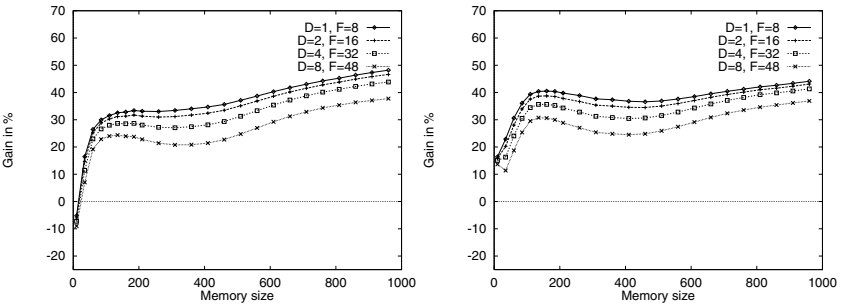


Fig. 4. Gain from using signatures, versus memory size, for different values of D. Non-temporal OODB to the left, temporal OODB to the right.

7.4 The Effect of P_A and P_{qm}

The value of P_{qm} is the fraction of the read queries that can benefit from signatures. Figure 6 illustrates the gain with different values of P_{qm}. As can be expected, a small value of P_{qm} results in negative gain in the case of non-temporal OODBs, i.e., in this case, storing and maintaining signatures in the OIDX reduces the average performance.

The value of P_A is the fraction of queries objects that are actual drops, the selectivity of the query. Only if the value of P_A is sufficiently low, will we be able to benefit from using signatures. Figure 7 shows that signatures will be beneficial even with a relatively large value of P_A.

8 Conclusions

We have in this paper described how signatures can be stored in the OIDX. As the OD is accessed on every object access in any case, there is no extra signature retrieval cost.

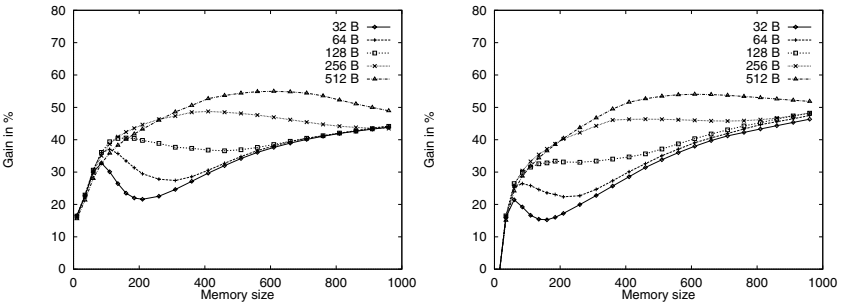


Fig. 5. Gain from using signatures, versus memory size, for different average object sizes. Non-temporal OODB to the left, temporal OODB to the right.

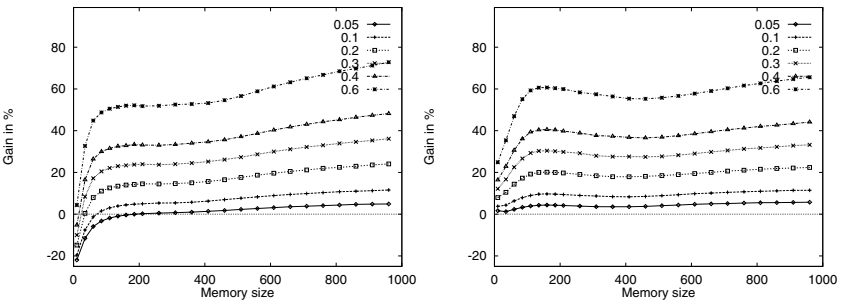


Fig. 6. Gain from using signatures, versus memory size, for different values of P_{qm}. Non-temporal OODB to the left, temporal OODB to the right.

In a traditional, non-versioned OODBs, maintaining signatures means that the OIDX needs to be updated every time an object is updated, but as the analysis shows, it will in most cases pay back, as less objects need to be retrieved.

Storing signatures in the OIDX is even more attractive for TOODBs. In TOODBs, the OIDX will have to be updated on every object update anyway, so in that case, the extra cost associated with signature maintenance is very low.

As showed in the analysis, substantial gain can be achieved by storing the signature in the OIDX. We have done the analysis with different system parameters, access patterns, and query patterns, and in most cases, storing the object signatures in the OIDX is beneficial. The typical gain is from 20 to 40%. Interesting to note is that the optimal signature size can be quite small.

The example analyzed in this paper is quite simple. A query for perfect match has a low complexity, and there is only limited room for improvement. The real benefit is available in queries where the signatures can be used to reduce the amount of data to be processed at subsequent stages of the query, resulting in larger amounts of data that

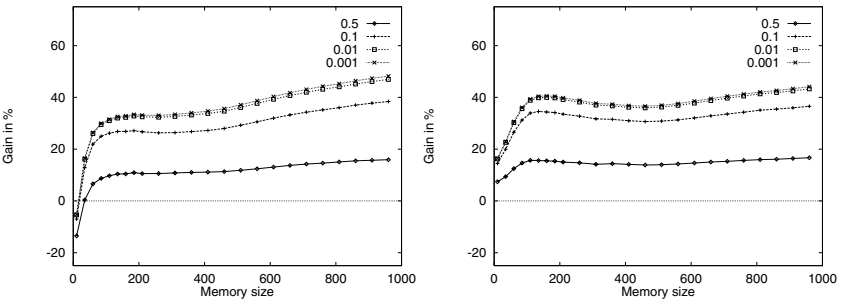


Fig. 7. Gain from using signatures, versus memory size, for different values of P_A. Non-temporal OODB to the left, temporal OODB to the right.

can be processed in main memory. This can speed up query processing several orders of magnitude.

Another interesting topic for further research is using signatures in the context of bitemporal TOODBs.

Acknowledgments

The author would like to thank Olav Sandstå for proofreading, and Kjell Bratbergsengen, who first introduced him to signatures.

References

1. E. Bertino and F. Marinaro. An evaluation of text access methods. In *Proceedings of the Twenty-Second Annual Hawaii International Conference on System Sciences, 1989. Vol.II: Software Track*, 1989. 367, 369
2. A. K. Bhide, A. Dan, and D. M. Dias. A simple analysis of the LRU buffer policy and its relationship to buffer warm-up transient. In *Proceedings of the Ninth International Conference on Data Engineering*, 1993. 373
3. M. Carey, D. DeWitt, J. Richardson, and E. Shekita. Object and file management in the EXODUS extensible database system. In *Proceedings of the 12th VLDB Conference, Kyoto, Japan, August 1986*, 1986. 375
4. A. Eickler, C. A. Gerlhof, and D. Kossmann. Performance evaluation of OID mapping techniques. In *Proceedings of the 21st VLDB Conference*, 1995. 376
5. R. Elmasri, G. T. J. Wuu, and V. Kouramajian. The time index and the monotonic B^+-tree. In A. U. Tansel, J. Clifford, S. K. Gadia, S. Jajodia, A. Segev, and R. Snodgrass, editors, *Temporal databases: theory, design and implementation*. The Benjamin/Cummings Publishing Company, Inc., 1993. 371
6. C. Faloutsos and R. Chan. Fast text access methods for optical and large magnetic disks: Designs and performance comparison. In *Proceedings of the 14th VLDB Conference*, 1988. 367, 369

7. Y. Ishikawa, H. Kitagawa, and N. Ohbo. Evaluation of signature files as set access facilities in OODBs. In *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, 1993. 369, 371
8. D. Lomet and B. Salzberg. Access methods for multiversion data. In *Proceedings of the 1989 ACM SIGMOD*, 1989. 371
9. K. Nørvåg and K. Bratbergsengen. Log-only temporal object storage. In *Proceedings of the 8th International Workshop on Database and Expert Systems Applications, DEXA'97*, 1997. 369
10. K. Nørvåg and K. Bratbergsengen. Optimizing OID indexing cost in temporal object-oriented database systems. In *Proceedings of the 5th International Conference on Foundations of Data Organization, FODO'98*, 1998. 372, 373

Author Index

Alagić, S. 14
Albrecht, J. 191

Bagai, R. 275
Brumen, B. 349
Budimac, Z. 168

Chandra, B. 98
Crowe, M. 218

Domajnko, T. 349
Dunne, P. 332
Dvorský, J. 75

Feng, J. 218

Gamage, C. 247
Gray, A. 332
Günzel, H. 191

Haase, O. 261
Hatzopoulos, M. 141
Henrich, A. 261
Heričko, M. 349
Herrmann, G. 1
Heuer, A. 288

Illner, R. 288
Ivanović, M. 168

Jun, W. 128
Jurič, M. B. 349

Kalinichenko, L. A. 317
Kapopoulos, D. G. 141
Kraken, A. 303
Kröger, J. 288

Laforest, F. 61
Lehner, W. 191
Leiwo, J. 247

Madria, S. K. 98
Maheshwari, S. 98
Maier, R. 232
Manolopoulos, Y. 85
Morzy, T. 179

Nardelli, E. 156
Norvag, K. 367

Pernul, G. 1
Płodzień, J. 303
Pokorný, J. 75
Popović, A. 168
Proietti, G. 156
Puuronen, S. 205

Röhm, A. W. 1
Rost, S. 288
Rozman, I. 349

Saake, G. 31, 113
Schwarz, K. 113
Snášel, V. 75

Tchounikine, A. 61
Terziyan, V. 205
Tran, N. 275
Tsymbal, A. 205
Türker, C. 31, 113
Tzouramanis, T. 85

Ursino, D. 46

Vassilakopoulos, M. 85

Welzer, T. 349
Wojciechowski, M. 179
Zakrzewicz, M. 179

Zheng, Y. 247
Živkovič, A. 349