LATVIJAS UNIVERSITĀTES
RAKSTI

672. SĒJUMS

Datorzinātne un informācijas tehnoloģijas

SCIENTIFIC PAPERS
UNIVERSITY OF LATVIA

VOLUME 672

Computer Science and Information Technologies

SCIENTIFIC PAPERS
UNIVERSITY OF LATVIA
VOLUME 672

Computer Science and Information Technologies

Databases and Information Systems

Edited by Janis Barzdins

Sixth International Baltic Conference
BalticDB&IS 2004
Riga, Latvia, June 6-9, 2004

LATVIJAS UNIVERSITĀTE

LATVIJAS UNIVERSITĀTES
RAKSTI
672. SĒJUMS

Datorzinātne un informācijas tehnoloģijas

Datu bāzes un informācijas sitēmas

Red. Jānis Bārzdiņš

Sestā Starptautiskā Baltijas konfernce
BalticDB&IS 2004
Rīga, Latvija, 2004. gada 6.–9. jūnijs

LATVIJAS UNIVERSITĀTE

UDK 004
Da 814

Visi krājumā ievietotie raksti ir recenzēti

ISSN 1407-2157
ISBN 9984-770-11-7

Table of Contents

Conference Committee

Advisory Committee
Janis Bubenko, Sweden
Arne Solvberg, Norway

Programme Co-Chairs
Janis Barzdins, Latvia
Albertas Caplinskas, Lithuania
Rusins Freivalds, Latvia

Organising Co-Chairs
Juris Borzovs, Latvia
Inara Opmane, Latvia

Co-ordinators
Ahto Kalja, Estonia
Saulius Maskeliunas, Lithuania

Programme Committee

Witold Abramowicz, Poland
Mikhail Auguston, USA
Janis Bicevskis, Latvia
Johann Eder, Austria
Hans-Dieter Ehrich, Germany
Jorgen Fischer Nilsson, Denmark
Janis Grundspenkis, Latvia
Remigijus Gustas, Sweden
Hele-Mai Haav, Estonia
Igor Kabashkin, Latvia
Leonid Kalinichenko, Russia
Ahto Kalja, Estonia
Audris Kalnins, Latvia
Marite Kirikova, Latvia
Patrick Lambrix, Sweden
Jozef M. Zurada, USA
Saulius Maskeliunas, Lithuania
Mihhail Matskin, Norway
Boris Novikov, Russia
Monika Oit, Estonia
Algirdas Pakstas, UK
Jaan Penjam, Estonia
Jaroslav Pokorny, Czech Republic
Gunter Saake, Germany
Klaus-Dieter Schewe, New Zealand
Jaak Tepandi, Estonia
Bernhard Thalheim, Germany
Enn Tyugu, Estonia
Olegas Vasilecas, Lithuania
Benkt Wangler, Sweden
Mudasser Wyne, USA
Arkady Zaslavsky, Australia

Additional Referees

Peter Ahlbrecht, Germany
Arne Ansper, Estonia
Per Backlund, Sweden
Dmitry Barashev, Russia
Asa Dahlstedt, Sweden
Rainer Eckstein, Germany
Silke Eckstein, Germany
Janis Eiduks, Latvia
Mohamed Medhat Gaber, Australia
Mika Hirvensalo, Finland
Hagen Hoepfner, Germany
Vaida Jakoniene, Sweden
Paulis Kikusts, Latvia
Markus Kirchberg, New Zealand
Christian Koncilia, Austria
Vahur Kotkas, Estonia
Algirdas Laukaitis, Lithuania
Marek Lehmann, Austria
Karl Neumann, Germany
Karlis Podnieks, Latvia
Oleg Proskurnin, Russia
Eike Schallehn, Germany
Ingo Schmitt, Germany
Asko Seeba, Estonia
Andrey Simanovsky, Russia
Eva Soderstrom, Sweden
Mattias Strand, Sweden
Alexei Tretiakov, New Zealand
Viljar Tulit, Estonia
Juris Viksna, Latvia

Local Organising Committee

Janis Barzdins
Janis Benefelds
Ansis Ataols Berzins
Janis Bicevskis
Uldis Bojars
Juris Borzovs, co-chair
Edgars Celms
Andrejs Dubrovskis
Anita Ermusa
Rusins Freivalds
Martins Gills
Ija Haritonova
Maksims Kravcevs
Davis Kulis
Lelde Lace
Ilvars Mizniks
Laila Niedrite
Inara Opmane, co-chair
Martins Opmanis
Raimonds Praude
Oksana Scegulnaja-Dubrovska
Darja Smite
Agnis Stibe
Maris Treimanis
Valdis Vitolins
Aleksandrs Zelenkovs
Janis Zuters

Conference Secretary
Edgars Celms, Latvia

Preface

Series of Baltic DB&IS biannual conferences was started in 1994. The idea has been suggested by Janis Bubenko, jr. and Arne Solvberg. It was the time when Baltic states returned to the European research Community. The first conference was held in Trakai (1994), then followed conferences in Tallinn (1996), Riga (1998), Vilnius (2000) and again in Tallinn (2002). And now you are welcome to Riga to the Sixth International Baltic Conference on Databases and Information Systems (Baltic DB&IS'2004).

Baltic DB&IS'2004 is organised by the Institute of Mathematics and Computer Science University of Latvia, Department of Computer Science (University of Latvia), Riga Information Technology Institute (Latvia) and the Institute of Mathematics and Informatics (Lithuania).

During this period, Baltic DB&IS conferences has become real international forums of high scientific criteria for academics and practitioners in the field of databases and modern information systems.

The Programme Committee of Baltic DB&IS'2004 consisted of 35 members from 14 countries. 59 papers from 17 countries were submitted for main conference and 27 papers from 5 countries for Doctoral Consortium. Each conference paper was reviewed by three referees, as a rule from different countries, and not from the country of the author of the paper. As a result, 35 papers were accepted for main conference. These papers are included in the conference proceedings.

The papers present original results in database technologies, database applications, data mining and warehousing, model transformations (MDA), specifications, ontologies, information systems and security.

The conference is preceded by one day Doctoral Consortium. Accepted DC papers are included in a separate volume of proceedings.

The Baltic DB&IS'2004 shares some sessions and other events with the 16th International Conference on Advanced Information Systems Engineering (CAiSE*04) which also is held in Riga, June 7-11.

We express our warmest thanks to all authors who contributed to the conference. We are grateful to members of the Programme Committee and the additional referees for carefully reviewing the submissions. We would also like to thank the organising team, especially Edgars Celms for the big technical work.

We express our very special thanks to our sponsors who have made this conference possible.

Last, but not least, we thank all participants who really made the conference.

Juris Borzovs
Organising Committee Co-Chair

Janis Barzdins
Programme Committee Co-Chair

DB TECHNOLOGIES

SCIENTIFIC PAPERS UNIVERSITY OF LATVIA. 2004. VOLUME 672, Datbases and Information Systems
13–30 P.

Generating and Optimising Views from Both as View Data Integration Rules

Edgar Jasper[1], Nerissa Tong[2], Peter McBrien[2], and Alexandra Poulovassilis[1]

[1] School of Computer Science and Information Systems, Birkbeck College, Univ. of London, {edgar,ap}@dcs.bbk.ac.uk
[2] Dept. of Computing, Imperial College, {nnyt98,pjm}@doc.ic.ac.uk

Abstract. This paper describes the generation and logical optimisation of views in the AutoMed heterogeneous data integration framework, which is based on the use of reversible schema transformation sequences called both as view (BAV) rules. We show how views can be generated from such sequences, for global as view (GAV), local as view (LAV) and GLAV query processing. We also present techniques for optimising these generated views, firstly by optimising the transformation sequences, and secondly by optimising the view definitions generated from them.

1 Introduction

Data integration is a process by which several databases, with associated **local schemas**, are integrated to form a single virtual database with an associated **global schema**. The two most common data integration approaches are **global as view** (**GAV**) (used in TSIMMIS [4], InterViso [19] and Garlic [18]), and **local as view** (**LAV**) (used in IM [9] and Agora [11]). In GAV, the constructs of a global schema are described as views over the local schemas. These view definitions are used to rewrite queries over a global schema into distributed queries over the local databases. In LAV, the constructs of the local schemas are defined as views over the global schema, and processing queries over the global schema involves rewriting queries using views [8].

Both LAV and GAV lack a certain degree of expressiveness. GAV is unable to fully capture data integration semantics where a source schema construct can be defined by a non-reversible function over global schema constructs. For example, if source schema attribute money is the sum of global schema attributes coins and notes, neither coins nor notes in the global schema can be defined by views over the source schema. Thus a query on the global schema asking for the sum of coins and notes cannot be answered even though the answer (money) is present in the source schema. In LAV, the attribute money can be defined by a view as the sum of global schema attributes coins and notes. Reversing the presence of the attributes, so that coins and notes are in the local schema and money in the global schema, leads to a situation which GAV can express but LAV cannot.

GLAV [5] is a variation of LAV that allows the head of the view definition rules to contain conjunctions of relations from a source schema as a natural join, and is thus able to capture situations where a non-reversible function is a natural-join between attributes. In [10] GLAV was extended to allow any source schema query in the head of the rule,

and hence is able to express the case where a single source schema is used to define the global schema constructs referenced in the body of the rule.

We have developed a richer integration framework which is based on the use of reversible sequences of primitive schema transformations, called transformation **pathways**. In [15] we showed how these pathways incorporate the semantics of GAV rule definitions and LAV rule definitions, and hence termed our approach **both as view** (**BAV**). We have implemented the BAV data integration approach within the AutoMed system (see `http://www.doc.ic.uk/automed`).

Since BAV integration is based on sequences of primitive schema transformations, it could be argued that the pathways resulting from BAV are likely to be more costly to reason with and process (*e.g.* for global query processing) than the corresponding LAV, GAV or GLAV view definitions would be. However, in Section 5 of this paper we show how BAV pathways are amenable to considerable simplification. Moreover, standard query optimisation techniques can be applied to the view definitions derived from BAV pathways.

The outline of this paper is as follows. Section 2 gives a review and examples of the BAV integration approach, and compares it with the GAV, LAV and GLAV approaches. Section 3 shows how view definitions can be generated from BAV pathways for GAV, LAV or GLAV query processing. Section 4 presents techniques for optimising these generated views, and Section 5 gives techniques for optimising the BAV pathways themselves. Section 6 gives our concluding remarks and directions of further work.

2 The BAV Integration Approach

In previous work (see `http://www.doc.ic.uk/automed`) we have developed a framework to support schema transformation and integration in heterogeneous database architectures. The framework consists of a low-level **hypergraph-based data model (HDM)** and a set of primitive schema transformations defined for this model. Higher-level data models and primitive schema transformations for them are defined in terms of this lower-level common data model.

In BAV, schemas are incrementally transformed by applying a sequence of primitive transformations $t_1, \ldots, t_r$, where each t_i adds, deletes or renames just one schema construct. Each add or delete transformation is accompanied by a query, expressed in the **intermediate query language** (**IQL**), specifying the extent of the new or deleted construct in terms of the rest of the constructs in the schema. All primitive transformations have an optional additional argument which specifies a constraint (also expressed in the IQL) on the current schema extension that must hold if the transformation is to be applied.

A **composite transformation** is a sequence of primitive transformations. We term the composite transformation defined for transforming schema S_1 to schema S_2 a transformation **pathway** $S_1 \rightarrow S_2$. All source schemas, intermediate schemas and global schemas, and the pathways between them are stored in AutoMed's metadata repository [1].

AutoMed supports a variety of methodologies for performing data integration and hence forming a **network** of pathways joining schemas together. For example, Figure 1 illustrates the integration of n local schemas, $LS_1, \ldots, LS_n$, into a global schema GS.

In order to integrate these n local schemas, each LS_i is first transformed into a "union" schema US_i. These n union schemas are syntactically identical, and this is asserted by creating a sequence of id transformation steps between each pair US_i and US_{i+1}, of the form id $(US_i:c, US_{i+1}:c)$ for each schema construct.

id is an additional type of primitive transformation, and the notation $US_i:c$ is used to denote construct c appearing in schema US_i. These id transformations are generated automatically by the AutoMed software. An arbitrary one of the US_i can then be selected for further transformation into a global schema GS. This is where constructs sourced from different local schemas can be combined together by unions, joins, outerjoins *etc*.

There may be information within a US_i which is not semantically derivable from the corresponding LS_i. This is asserted by means of extend transformation steps within the pathway $LS_i \rightarrow US_i$. Conversely, not all of the information within a local schema LS_i need be transferred into US_i, and this is asserted by means of contract transformation steps within $LS_i \rightarrow US_i$. These extend and contract transformations behave in the same way as add and delete, respectively, except that they indicate that only partial information can be derived about the new or deleted construct. Rather than a single query, they take a pair of queries which specify a lower and upper bound on the extent of the new or deleted construct. The lower bound query may be the constant Void if no lower bound can be specified, and the upper bound query may be the constant Any if no upper bound can be specified.

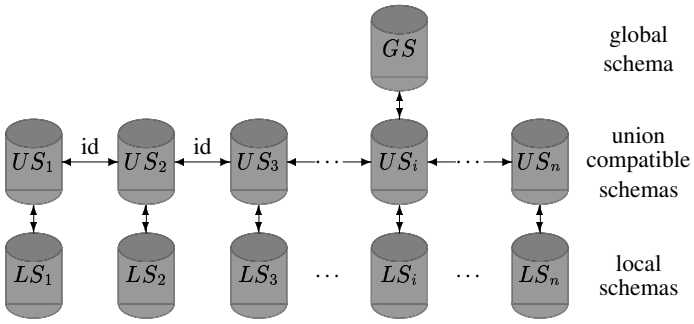


Fig. 1. A general AutoMed Integration

Each primitive transformation t has an **automatically derivable** reverse transformation $\overline{t}$. In particular, each add or extend transformation is reversed by a delete or contract transformation with the same arguments, and vice versa, while each rename or id transformation is reversed by another rename or id transformation with the two arguments swapped. This holds for the primitive transformations of **any** modelling language defined in AutoMed. In [12] we show how this reversibility of schema transformations allows automatic data query translation between schemas.

In [13] we described how our framework can be applied to different high-level modelling languages such as relational, ER and UML, and more recently we have extended AutoMed to also support semi-structured data models (flat file, XML, RDF). For our examples in this paper we will use a simplified relational data model. However, we stress

that the techniques that we describe here are equally applicable to any data modelling language supported by AutoMed.

In our simple relational model, there are two kinds of schema construct: Rel and Att (see [13] for an encoding of a richer relational data model, including the modelling of constraints).

The extent of a Rel construct $\langle\langle R \rangle\rangle$ is the projection of the relation R onto its primary key attributes $k_1, ..., k_n$. The extent of each Att construct $\langle\langle R, a \rangle\rangle$ where a is an attribute (key or non-key) is the projection of relation R onto $k_1, ..., k_n, a$. For example, a relation student(id,sex,dname) would be modelled by a Rel construct $\langle\langle \mathsf{student} \rangle\rangle$, and three Att constructs $\langle\langle \mathsf{student}, \mathsf{id} \rangle\rangle$, $\langle\langle \mathsf{student}, \mathsf{sex} \rangle\rangle$ and $\langle\langle \mathsf{student}, \mathsf{dname} \rangle\rangle$.

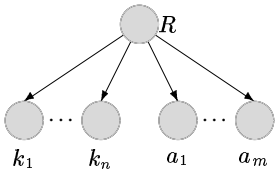


Fig. 2. A simple relational data model

Once the constructs of modelling language $\mathcal{M}$ have been defined in terms of the HDM (via the API of AutoMed's metadata repository [1]), a set of primitive schema transformations for $\mathcal{M}$ are automatically available. For the simple relational model above, these would be as follows:

- addRel($\langle\langle R \rangle\rangle, q$) adds to the schema a new relation R. The query q specifies the set of primary key values in the extent of R in terms of the already existing schema constructs.
- addAtt($\langle\langle R, a \rangle\rangle, q$) adds to the schema an attribute a (key or non-key) for relation R. The query q specifies the extent of the binary relationship between the primary key attribute(s) of R and this new attribute a in terms of the already existing schema constructs.
- deleteRel($\langle\langle R \rangle\rangle, q$) deletes from the schema the relation R (provided all its attributes have first been deleted). The query q specifies how the extent of R can be restored from the remaining schema constructs.
- deleteAtt($\langle\langle R, a \rangle\rangle, q$) deletes from the schema attribute a of relation R. The query q specifies how the extent of the binary relationship between the primary key attribute(s) of R and a can be restored from the remaining schema constructs.
- renameRel($\langle\langle R \rangle\rangle, \langle\langle R' \rangle\rangle$) renames the relation R to R' in the schema.
- renameAtt($\langle\langle R, a \rangle\rangle, \langle\langle R, a' \rangle\rangle$) renames the attribute a of R to a'.

There is also a set of extendRel, extendAtt, contractRel and contractAtt primitive transformations.

2.1 An Example Integration

Figure 3 gives some specific schemas to illustrate the integration approach of Figure 1. Primary key attributes are underlined, foreign key attributes are in italics and nullable attributes are suffixed by a question mark.

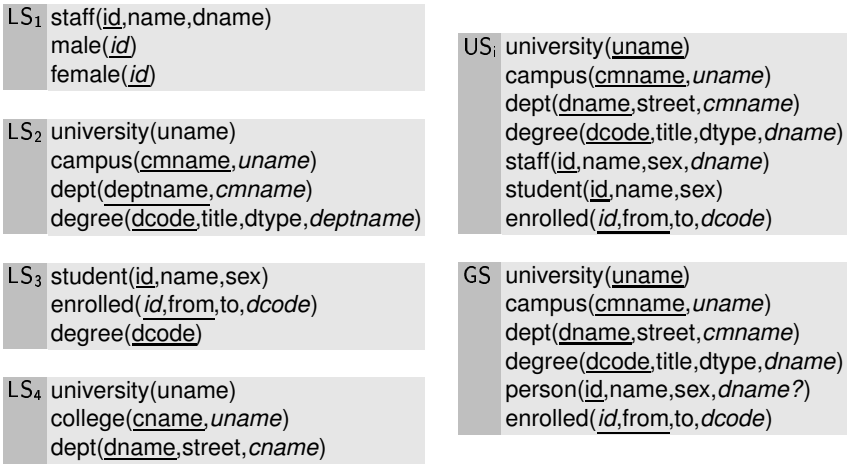


Fig. 3. Example schemas

In Example 1, transformations t_1–t_5 use a composite transformation extendTable to state that the tables student, enrolled, university, campus and degree in US_1 cannot be derived from LS_1. The definition of extendTable is:

$$\mathsf{extendTable}(\langle\langle R, a_1, \ldots, a_n\rangle\rangle) = \mathsf{extendRel}(\langle\langle R\rangle\rangle, \mathsf{Void}, \mathsf{Any});$$
$$\mathsf{extendAtt}(\langle\langle R, a_1\rangle\rangle, \mathsf{Void}, \mathsf{Any}); \ldots; \mathsf{extendAtt}(\langle\langle R, a_n\rangle\rangle, \mathsf{Void}, \mathsf{Any})$$

Then transformations t_6–t_9 use the dname attribute of staff to derive the dept table in US_1, and use extend transformations for the two attributes street and uname that cannot be derived from LS_1. Finally, in t_{10}–t_{14} the male and female relations of LS_1 are restructured into the single sex attribute of staff.

The queries accompanying the add and delete transformations are expressed in our IQL intermediate query language. In IQL, ++ is the bag union operator and the construct $[e \mid Q_1; \ldots Q_n]$ is a **comprehension** [2]. The expressions Q_1 to Q_n are termed **qualifiers**, each qualifier being either a **filter** or a **generator**. A filter is a boolean-valued expression. A generator has syntax $p \leftarrow c$ where p is a **pattern** and c is a bag-valued expression. In IQL, the patterns p are restricted to be single variables or tuples of variables.

Example 1 Pathway $\mathsf{LS}_1 \rightarrow \mathsf{US}_1$

t_1 extendTable(⟨⟨student, id, name, sex⟩⟩)
t_2 extendTable(⟨⟨university, uname⟩⟩)
t_3 extendTable(⟨⟨campus, cmname, uname⟩⟩)
t_4 extendTable(⟨⟨degree, dcode, title, dtype, dname⟩⟩)
t_5 extendTable(⟨⟨enrolled, id, from, to, dcode⟩⟩)
t_6 addRel(⟨⟨dept⟩⟩, [x | (y, x) ← ⟨⟨staff, dname⟩⟩])
t_7 addAtt(⟨⟨dept, dname⟩⟩, [(x, x) | x ← ⟨⟨dept⟩⟩])
t_8 extendAtt(⟨⟨dept, street⟩⟩, Void, Any)
t_9 extendAtt(⟨⟨dept, uname⟩⟩, Void, Any)

t_{10} addAtt(⟨⟨staff, sex⟩⟩, [(x, 'M') | x ← ⟨⟨male⟩⟩] ++ [(x, 'F') | x ← ⟨⟨female⟩⟩])
t_{11} deleteAtt(⟨⟨male, id⟩⟩, [(x, x) | x ← ⟨⟨male⟩⟩])
t_{12} deleteRel(⟨⟨male⟩⟩, [x | (x, 'M') ← ⟨⟨staff, sex⟩⟩])
t_{13} deleteAtt(⟨⟨female, id⟩⟩, [(x, x) | x ← ⟨⟨female⟩⟩])
t_{14} deleteRel(⟨⟨female⟩⟩, [x | (x, 'F') ← ⟨⟨staff, sex⟩⟩])

The pathway $LS_2 \rightarrow US_2$ contains extend steps to add the missing staff, student, and enrolled tables. It then renames deptname, and adds the missing attributes of dept:

Example 2 Pathway $LS_2 \rightarrow US_2$
t_{15} extendTable(⟨⟨student, id, name, sex⟩⟩)
t_{16} extendTable(⟨⟨staff, id, name, sex, dname⟩⟩)
t_{17} extendTable(⟨⟨enrolled, id, from, to, dcode⟩⟩)
t_{18} renameAtt(⟨⟨dept, deptname⟩⟩, ⟨⟨dept, dname⟩⟩)
t_{19} renameAtt(⟨⟨degree, deptname⟩⟩, ⟨⟨degree, dname⟩⟩)
t_{20} extendAtt(⟨⟨dept, street⟩⟩, Void, Any)
t_{21} extendAtt(⟨⟨dept, uname⟩⟩, Void, Any)

The pathway $LS_3 \rightarrow US_3$ contains a sequence of extend steps for its missing information. The pathway $LS_4 \rightarrow US_4$ creates in t_{22} a new attribute ⟨⟨dept, uname⟩⟩ by joining the dept and college relations, and then deletes in t_{23}–t_{25} the college table that can be recovered from the remaining ⟨⟨dept, cname⟩⟩ attribute. Transformation t_{26} is unable to put any restriction on the values of ⟨⟨dept, cname⟩⟩, since that association cannot be recovered from the global schema. Transformations t_{27}–t_{31} then perform the logical inverse of t_{22}–t_{26} to partially extract the campus table from the direct association between departments and universities represented by ⟨⟨dept, uname⟩⟩.

Example 3 Pathway $LS_4 \rightarrow US_4$
t_{22} addAtt(⟨⟨dept, uname⟩⟩,
    [(x, y) | (x, z) ← ⟨⟨dept, cname⟩⟩; (z, y) ← ⟨⟨college, uname⟩⟩])
t_{23} deleteAtt(⟨⟨college, uname⟩⟩,
    [(x, y) | (z, x) ← ⟨⟨dept, cname⟩⟩; (z, y) ← ⟨⟨dept, uname⟩⟩])
t_{24} deleteAtt(⟨⟨college, cname⟩⟩, [(x, x) | x ← ⟨⟨college⟩⟩])
t_{25} deleteRel(⟨⟨college⟩⟩, [y | (x, y) ← ⟨⟨dept, cname⟩⟩])
t_{26} contractAtt(⟨⟨dept, cname⟩⟩, Void, Any)
t_{27} extendAtt(⟨⟨dept, cmname⟩⟩, Void, Any)
t_{28} addRel(⟨⟨campus⟩⟩, [y | (x, y) ← ⟨⟨dept, cmname⟩⟩])
t_{29} addAtt(⟨⟨campus, cmname⟩⟩, [(x, x) | x ← ⟨⟨campus⟩⟩])
t_{30} addAtt(⟨⟨campus, uname⟩⟩,
    [(x, y) | (z, x) ← ⟨⟨dept, cmname⟩⟩; (z, y) ← ⟨⟨dept, uname⟩⟩])
t_{31} delAtt(⟨⟨dept, uname⟩⟩,
    [(x, y) | (x, z) ← ⟨⟨dept, cmname⟩⟩; (z, y) ← ⟨⟨campus, uname⟩⟩])
t_{32} extendTable(⟨⟨student, id, name, sex⟩⟩)
t_{33} extendTable(⟨⟨staff, id, name, sex, dname⟩⟩)
t_{34} extendTable(⟨⟨enrolled, id, from, to, dcode⟩⟩)

Finally, we list in Example 4 the pathway from the union schema US_1 to the global schema GS. The pathway from US_2, US_3 or US_4 would be identical.

Example 4 Pathway $\mathsf{US}_1 \rightarrow \mathsf{GS}$

t_{35} addRel(⟨⟨person⟩⟩, ⟨⟨staff⟩⟩ ++ [x | x ← ⟨⟨student⟩⟩; not (member x ⟨⟨staff⟩⟩)])
t_{36} addAtt(⟨⟨person, id⟩⟩, ⟨⟨staff, id⟩⟩ ++
[(x, y) | (x, y) ← ⟨⟨student, id⟩⟩; not (member x ⟨⟨staff⟩⟩)])
t_{37} addAtt(⟨⟨person, name⟩⟩, ⟨⟨staff, name⟩⟩ ++
[(x, y) | (x, y) ← ⟨⟨student, name⟩⟩; not (member x ⟨⟨staff⟩⟩)])
t_{38} addAtt(⟨⟨person, sex⟩⟩, ⟨⟨staff, sex⟩⟩++
[(x, y) | (x, y) ← ⟨⟨student, sex⟩⟩; not (member x ⟨⟨staff⟩⟩)])
t_{39} addAtt(⟨⟨person, dname⟩⟩, ⟨⟨staff, dname⟩⟩)
t_{40} contractAtt(⟨⟨student, id⟩⟩, [(x, y) | (x, y) ← ⟨⟨person, id⟩⟩;
not (member x ⟨⟨staff⟩⟩)], [(x, y) | (x, y) ← ⟨⟨person, id⟩⟩])
t_{41} contractAtt(⟨⟨student, name⟩⟩, [(x, y) | (x, y) ← ⟨⟨person, name⟩⟩;
not (member x ⟨⟨staff⟩⟩)], [(x, y) | (x, y) ← ⟨⟨person, name⟩⟩])
t_{42} contractAtt(⟨⟨student, sex⟩⟩, [(x, y) | (x, y) ← ⟨⟨person, sex⟩⟩;
not (member x ⟨⟨staff⟩⟩)], [(x, y) | (x, y) ← ⟨⟨person, sex⟩⟩])
t_{43} contractRel(⟨⟨student⟩⟩), [x | x ← ⟨⟨person⟩⟩; not (member x ⟨⟨staff⟩⟩)],
[x | x ← ⟨⟨person⟩⟩])
t_{44} deleteAtt(⟨⟨staff, id⟩⟩, [(x, y) | (x, y) ← ⟨⟨person, id⟩⟩; member x ⟨⟨staff⟩⟩])
t_{45} deleteAtt(⟨⟨staff, name⟩⟩, [(x, y) | (x, y) ← ⟨⟨person, name⟩⟩; member x ⟨⟨staff⟩⟩])
t_{46} deleteAtt(⟨⟨staff, sex⟩⟩, [(x, y) | (x, y) ← ⟨⟨person, sex⟩⟩; member x ⟨⟨staff⟩⟩])
t_{47} deleteAtt(⟨⟨staff, dname⟩⟩, ⟨⟨person, dname⟩⟩)
t_{48} deleteRel(⟨⟨staff⟩⟩, [x | (x, y) ← ⟨⟨person, dname⟩⟩])

We assume in this example integration that a person may be both a member of staff and a student. For such people, their information in the staff table is preferred for propagation to the global person table in steps t_{35}–t_{38} above. Thus, there is not sufficient information in the global schema to totally derive the student table, and only contract statements can be given in steps t_{40}–t_{43}, where as a lower bound we know all persons not in the staff table are students, but as an upper bound know that all persons could be in student (if it were the case that all staff members were former students). Conversely, there is sufficient information to totally derive the staff table.

2.2 Comparison of BAV with GAV, LAV and GLAV

We see from the above example that the add and extend steps in the transformation pathways from the local schemas to the global schema correspond to GAV, since it is these steps that are incrementally defining global constructs in terms of local ones. Similarly, it is the delete and contract steps in the transformation pathways from the local schemas to the global schema that correspond to LAV, since it is these steps that are incrementally defining local constructs in terms of global ones. We will see in Section 3 how these pathways can be traversed to derive GAV and LAV views.

If a GAV view is derived from solely add steps it will be *exact* in the terminology of [7]. If, in addition, it is derived from one or more extend steps using their lower-bound

(upper-bound) queries, then the GAV view will be *sound* (*complete*) in the terminology of [7]. Similarly, if a LAV view is derived solely from delete steps it will be exact. If, in addition, it is derived from one or more contract steps using their lower-bound (upper-bound) queries, then the LAV view will be *complete* (*sound*) in the terminology of [7]. For example, in pathway $\mathsf{US}_1 \rightarrow \mathsf{GS}$ above, we could enhance t_{43} above to:
$\mathsf{contractRel}(\langle\langle\mathsf{student}\rangle\rangle, [\mathsf{x} \mid \mathsf{x} \leftarrow \langle\langle\mathsf{person}\rangle\rangle; \mathsf{not}\ (\mathsf{member}\ \mathsf{x}\ \langle\langle\mathsf{staff}\rangle\rangle)], \langle\langle\mathsf{person}\rangle\rangle])$
asserting that $\langle\langle\mathsf{student}\rangle\rangle$ contains the set of people who are not staff (completeness) and is contained by the whole set of people (soundness).

As we discussed in the introduction, BAV is a more expressive data integration language than LAV, GAV or GLAV, since it allows for the expression of mappings in both directions, and since it is not limited on how many source schemas are associated by a mapping. Indeed, in the context of peer-to-peer integration, [3] has suggested using GLAV rules in both directions in a similar manner to BAV, in order to overcome weaknesses of using GLAV alone.

As discussed in [14, 15], a further advantage of BAV over GAV and LAV is that it readily supports the evolution of both global and local schemas, by allowing pathways and schemas to be incrementally modified as opposed to having to be regenerated.

A further difference between BAV and GAV, LAV or GLAV (including the approach of using GLAV in each direction of [3]) is that statements about the relationships between global and local schemas are made at a finer level of detail, *i.e.* at the level of individual attributes as opposed to entire tables. So we can assert exact knowledge about some attributes of a table, and sound or complete knowledge about other attributes. We are also able to introduce intermediate constructs in the mapping, such as in $\mathsf{LS}_4 \rightarrow \mathsf{US}_4$.

3 Generating Views

We now present a general technique for generating GAV, LAV and GLAV view definitions from a BAV pathway. This ability to generate any of these kinds of view definitions from a single BAV pathway means that we can select a query processing technique that can vary between queries as appropriate.

To define a construct c in S_x in terms of the constructs in schema S_y, we consider in turn the transformations of $S_x \rightarrow S_y$. The only transformations that are significant are those that delete, contract or rename a construct[1]. These transformations are significant because the current view definitions may query constructs that no longer exist after such a transformation. Each of these types of transformation is handled as follows if it is encountered during the traversal of $S_x \rightarrow S_y$:

- delete: This has an associated query which shows how to reconstruct the extent of the construct being deleted. Any occurrence of the deleted construct within the current view definitions is replaced by this query.
- contract: Any occurrence of the contracted construct within the current view definitions is replaced by either the lower-bound or the upper-bound query accompa-

[1] Note that this is equivalent to considering the add, extend and rename steps in the reverse $S_y \rightarrow S_x$

nying this transformation step, depending on whether sound or complete views are required.
- rename: All references to the old construct in the current view definitions are replaced by references to the new construct.

3.1 Generating GAV Views

To generate the set of GAV views for a global schema, the pathways from it to each local schema are retrieved from AutoMed's metadata repository. For some part of the start of their length these pathways may be the same, as may be seen from the tree structure of Figure 1. Each node of this transformations tree is a schema (global, intermediate or local) linked to its neighbours by a single transformation step. View definitions for each global schema construct are derived by traversing the tree from top to bottom. Initially, each construct's view definition is just the construct itself. Each node in the tree is then visited in a downwards direction, and delete, contract and rename transformations are handled as described above. In particular, if a contract transformation step is encountered, any occurrence of the contracted construct within the current GAV view definitions is replaced by the lower-bound query accompanying this transformation step (so that sound GAV views will be generated).

At some points the tree may branch. When this happens, constructs of the parent schema are semantically identical to constructs that have the same scheme within the child schemas. The possibility of using all paths is retained within the view definitions by replacing each construct of the parent schema by a disjunction (OR) of the corresponding constructs in the child schemas.

The tree is traversed in this fashion from the root to the leaves until all the nodes are visited. The resulting view definitions are the GAV definitions for the global schema constructs over the local schemas. Referring again to the example of Section 2.1, consider the construct $\mathsf{GS}:\langle\langle \mathsf{person}, \mathsf{sex}\rangle\rangle$ in the global schema. The pathway $\mathsf{GS} \rightarrow \mathsf{US}_1$ would be processed first (*i.e.* the reverse of the pathway $\mathsf{US}_1 \rightarrow \mathsf{GS}$ listed in Section 2.1). The only significant transformation is $\overline{t_{38}}$ that deletes $\langle\langle \mathsf{person}, \mathsf{sex}\rangle\rangle$, resulting in an intermediate view definition:

$\mathsf{GS}:\langle\langle \mathsf{person}, \mathsf{sex}\rangle\rangle$:- $\mathsf{US}_1:\langle\langle \mathsf{staff}, \mathsf{sex}\rangle\rangle$ ++
$[(x,y) \mid (x,y) \leftarrow \mathsf{US}_1:\langle\langle \mathsf{student}, \mathsf{sex}\rangle\rangle; \mathsf{not}\ (\mathsf{member}\ x\ \mathsf{US}_1:\langle\langle \mathsf{staff}\rangle\rangle)]$

at one copy, US_1, of the four union schemas. Traversing the pathways $\mathsf{US}_1 \rightarrow \mathsf{LS}_1$ and $\mathsf{US}_1 \rightarrow \mathsf{US}_2$, we replace the body of the view definition with:

$([(x,\text{'M'}) \mid x \leftarrow \mathsf{LS}_1:\langle\langle \mathsf{male}\rangle\rangle] ++ [(x,\text{'F'}) \mid x \leftarrow \mathsf{LS}_1:\langle\langle \mathsf{female}\rangle\rangle]\ \mathsf{OR}$
$\mathsf{US}_2:\langle\langle \mathsf{staff}, \mathsf{sex}\rangle\rangle)$
$++ ([(x,y) \mid (x,y) \leftarrow \mathsf{Void}\ \mathsf{OR}\ \mathsf{US}_2:\langle\langle \mathsf{student}, \mathsf{sex}\rangle\rangle;$
$\mathsf{not}\ (\mathsf{member}\ x\ (\mathsf{LS}_1:\langle\langle \mathsf{staff}\rangle\rangle\ \mathsf{OR}\ \mathsf{US}_2:\langle\langle \mathsf{staff}\rangle\rangle))])$

Traversing next $\mathsf{US}_2 \rightarrow \mathsf{LS}_2$ and $\mathsf{US}_2 \rightarrow \mathsf{US}_3$, we get:

$([(x,\text{'M'}) \mid x \leftarrow \mathsf{LS}_1:\langle\langle \mathsf{male}\rangle\rangle] ++ [(x,\text{'F'}) \mid x \leftarrow \mathsf{LS}_1:\langle\langle \mathsf{female}\rangle\rangle])\ \mathsf{OR}$
$\mathsf{Void}\ \mathsf{OR}\ \mathsf{US}_3:\langle\langle \mathsf{staff}, \mathsf{sex}\rangle\rangle)$
$++ ([(x,y) \mid (x,y) \leftarrow \mathsf{Void}\ \mathsf{OR}\ \mathsf{Void}\ \mathsf{OR}\ \mathsf{US}_3:\langle\langle \mathsf{student}, \mathsf{sex}\rangle\rangle;$
$\mathsf{not}\ (\mathsf{member}\ x\ (\mathsf{LS}_1:\langle\langle \mathsf{staff}\rangle\rangle\ \mathsf{OR}\ \mathsf{Void}\ \mathsf{OR}\ \mathsf{US}_3:\langle\langle \mathsf{staff}\rangle\rangle))])$

Continuing with $\mathsf{US}_3 \rightarrow \mathsf{LS}_3$, $\mathsf{US}_3 \rightarrow \mathsf{US}_4$ and finally $\mathsf{US}_4 \rightarrow \mathsf{LS}_4$, we obtain the view definition:

GS : ⟨⟨person, sex⟩⟩ :-
([(x, 'M') | x ← LS$_1$: ⟨⟨male⟩⟩] ++ [(x, 'F') | x ← LS$_1$: ⟨⟨female⟩⟩])OR
Void OR Void OR Void)
++ ([(x, y) | (x, y) ← Void OR Void OR LS$_3$: ⟨⟨student, sex⟩⟩ OR Void;
not (member x (LS$_1$: ⟨⟨staff⟩⟩ OR Void OR Void OR Void))])

Such view derivations can be substituted into any query posed on a global schema in order to obtain an equivalent query distributed over the local schemas — this is the GAV approach to global query processing, which is what the AutoMed implementation currently supports. Section 4 will justify how this view definition can be simplified further.

3.2 Generating LAV Views

LAV views are derived similarly: the pathway from a local schema to the global schema is again retrieved from the metadata repository and is processed as above to derive the view definitions, except that it is the local schema end of the pathway that is now taken as the root of the tree. The derivation of LAV views is simpler because there is now only a single pathway being processed, with no branching. Also, if a contract transformation step is encountered, any occurrence of the contracted construct within the current LAV view definitions is replaced by the upper-bound query accompanying this transformation step (so that sound LAV views will be generated).

For example, to generate a LAV definition of LS$_1$: ⟨⟨male⟩⟩, we inspect the pathway t_1,..,t_{14},t_{35},..,t_{48}. The transformation t_{12} deletes ⟨⟨male⟩⟩, and therefore we have an intermediate view definition on US$_1$:
LS$_1$: ⟨⟨male⟩⟩ :- [x | (x, 'M') ← US$_1$: ⟨⟨staff, sex⟩⟩]

Then ⟨⟨staff, sex⟩⟩ construct is deleted by t_{46}, which substitutes(x, y) ← ⟨⟨staff, sex⟩⟩ with (x, y) ← ⟨⟨person, sex⟩⟩; member x ⟨⟨staff⟩⟩, and the ⟨⟨staff⟩⟩ construct in this query is deleted by t_{40} giving a final LAV rule:
LS$_1$: ⟨⟨male⟩⟩ :-
[x | (x, 'M') ← GS : ⟨⟨person, sex⟩⟩; member x [x | (x, y) ← GS : ⟨⟨person, dname⟩⟩]]

3.3 Generating GLAV Views

First, it should be noted that GLAV view definitions will include all the LAV view definitions, and all the GAV view definitions where the body of the rule is a query that matches the conditions required for the GLAV query processing system in use (which in [10] would be queries over a single source). In addition, we inspect now all the add and extend transformations of the pathway that would be ignored by LAV view generation, and for each one use the query to form the head of a new GLAV rule.

For example, in LS$_4$ → US$_4$, the query in t_{22} gives a new view rule head:
[(x, y) | (x, z) ← LS$_4$: ⟨⟨dept, cname⟩⟩; (z, y) ← LS$_4$: ⟨⟨college, uname⟩⟩] which is defined by ⟨⟨dept, uname⟩⟩ at this stage. We then use our standard algorithm on construct ⟨⟨dept, uname⟩⟩, detect that it is deleted in t_{31}, and hence replace it with the query from t_{31} to result in the GLAV rule:
[(x, y) | (x, z) ← LS$_4$: ⟨⟨dept, cname⟩⟩; (z, y) ← LS$_4$: ⟨⟨college, uname⟩⟩] :-
[(x, z) | GS : ⟨⟨dept, cmname⟩⟩; (z, y) ← GS : ⟨⟨campus, uname⟩⟩]

Note however that the BAV integration would still hold if LS_4 were fragmented, with campus and departments held on separate sources, whereas GLAV would cease to be valid in this situation.

4 Logical Optimisation of the Generated Views

The view definitions generated by the process described above can be simplified by a process of logical optimisation, where redundant parts of the query are removed. This saves later work for the query optimiser, when these definitions are substituted into specific global queries for query processing. It also generates views that are similar to the views that would have been specified directly in a GAV, LAV or GLAV framework.

4.1 The OR Operator and Void

The Void value represents a construct that is unobtainable from a data source. We thus define $\mathsf{e}\ \mathsf{OR}\ \mathsf{Void} = \mathsf{Void}\ \mathsf{OR}\ \mathsf{e} = \mathsf{e}$ for any IQL expression e. Applying this simplification to the GAV view definition derived in Section 3.1 results in:

$\mathsf{GS} : \langle\langle \mathsf{person}, \mathsf{sex} \rangle\rangle$:-
($[(\mathsf{x}, \text{'M'}) \mid \mathsf{x} \leftarrow \mathsf{LS}_1 : \langle\langle \mathsf{male} \rangle\rangle] ++ [(\mathsf{x}, \text{'F'}) \mid \mathsf{x} \leftarrow \mathsf{LS}_1 : \langle\langle \mathsf{female} \rangle\rangle]$))
++ ($[(\mathsf{x}, \mathsf{y}) \mid (\mathsf{x}, \mathsf{y}) \leftarrow \mathsf{LS}_3 : \langle\langle \mathsf{student}, \mathsf{sex} \rangle\rangle; \mathsf{not}\ (\mathsf{member}\ \mathsf{x}\ (\mathsf{LS}_1 : \langle\langle \mathsf{staff} \rangle\rangle))]$)

It may be the case that two data sources supply information for a single schema construct. For example, the global schema attribute $\langle\langle \mathsf{university}, \mathsf{uname} \rangle\rangle$ has the GAV view definition:

$\mathsf{GS} : \langle\langle \mathsf{university}, \mathsf{uname} \rangle\rangle$:- $\mathsf{LS}_2 : \langle\langle \mathsf{university}, \mathsf{uname} \rangle\rangle\ \mathsf{OR}\ \mathsf{LS}_4 : \langle\langle \mathsf{university}, \mathsf{uname} \rangle\rangle$

which expresses the fact that either LS_2 or LS_4 can be used to extract information about university names. This leads to several possibilities for operational semantics that may be used for the OR operator:

1. **ident** semantics would choose one of the expressions to evaluate, since the integration rules specify that they are the same. This may be defined by the rule $\mathsf{e1}\ \mathsf{OR}\ \mathsf{e2} = \mathsf{e1} = \mathsf{e2}$.
2. **intersect** semantics would determine that a value should be returned only if it is present in all data sources, defined by $\mathsf{e1}\ \mathsf{OR}\ \mathsf{e2} = \mathsf{intersect}\ \mathsf{e1}\ \mathsf{e2}$.
3. **append** semantics would determine that all values in all data sources should be returned, defined by $\mathsf{e1}\ \mathsf{OR}\ \mathsf{e2} = \mathsf{e1} ++ \mathsf{e2}$.
4. **union** semantics would determine that one copy of a value should be returned if present in any data source, defined by $\mathsf{e1}\ \mathsf{OR}\ \mathsf{e2} = \mathsf{distinct}\ (\mathsf{e1} ++ \mathsf{e2})$.

Option (1) is that which should be used if it is known that the data sources obey the semantics specified by the data integration rules *i.e.* that their extents are identical and there are no distributed data integrity violations. In this circumstance, the OR operator may also be used during distributed data integrity checking, where both expressions are evaluated, and the results compared to determine if the data sources contain consistent data.

Options (2)–(4) provide different mechanisms for handling situations where the data sources are possibly inconsistent, and thus may not share information that they should

share. Option (3) provides a result that may be used to derive Options (2) and (4), and therefore is the default semantics provided by the AutoMed's view generation algorithm. Note also that Option (4) gives the same result as Option (1) if the data sources are identical.

4.2 Other IQL Operators

The AutoMed intermediate query language IQL supports several primitive operators for manipulating lists. The list **append** operator, $++$, concatenates two lists together. The **distinct** operator removes duplicates from a list. The **monus** operator $--$ subtracts each instance of the second list from the first. For example, $[1,\ 2,\ 3,\ 2,\ 4] -- [4,\ 4,\ 2,\ 1] = [3,\ 2]$. The **fold** operator applies a given function f to each element of a list and then 'folds' a binary operator op into the resulting values, and is defined as follows:

fold f op e [ ] = e

fold f op e [x] = f x

fold f op e (b1 ++ b2) = (fold f op e b1) op (fold f op e b2)

Other IQL list manipulation operators may be defined using fold together with the usual set of built-in operators and also the support of lambda abstractions. For example, the IQL functions sum and count are equivalent to the SQL SUM and COUNT aggregation functions and are defined respectively as sum = fold (id) (+) 0 and count = fold (lambda x.1) (+) 0.

The function flatmap applies a list-valued function f to each member of a list b and is defined as flatmap f b = fold f (++) [] b. flatmap can in turn be used to define selection, projection, join and, more generally, the comprehension syntax used in the view definitions of the previous section. For example, the list comprehension [x | x ← ⟨⟨student⟩⟩; not (member x ⟨⟨staff⟩⟩)] translates to:

flatmap (lambda x.if (not (member x ⟨⟨staff⟩⟩)) then [x] else []) ⟨⟨student⟩⟩

Optimisations for fold apply to all the operators defined in terms of it. Regarding the view definitions generated from BAV pathways there are two particular optimisations that can be applied to them. First, any instances of fold applied to Void can be simplied by treating Void as identical to the empty bag, so that fold f op e Void = e for any f, op, e. Second, due to the step-wise specification of our schema transformations, **loop fusion** may be applicable. This replaces two successive iterations over a collection by one iteration provided the operators in question satisfy certain algebraic properties. A simple instance of loop fusion is the standard relational query optimisation $\pi_A(\pi_B(R)) = \pi_{A,B}(R)$. Loop fusion does not arise in the schema integration example of Section 2.1 but consider the following fragment of an AutoMed pathway. This first joins two schemes $\langle\langle R, a\rangle\rangle$ and $\langle\langle R, b\rangle\rangle$, creating an intermediate relation $\langle\langle I\rangle\rangle$, and then projects onto the a and b attributes, creating a relation $\langle\langle V\rangle\rangle$, and finally deletes $\langle\langle I\rangle\rangle$:

addRel($\langle\langle I\rangle\rangle$, $[(x,y,z) \mid (x,y) \leftarrow \langle\langle R,a\rangle\rangle; (x,z) \leftarrow \langle\langle R,b\rangle\rangle]$)

addRel($\langle\langle V\rangle\rangle$, $[(y,z) \mid (x,y,z) \leftarrow \langle\langle I\rangle\rangle]$)

deleteRel($\langle\langle I\rangle\rangle$, $[(x,y,z) \mid (x,y) \leftarrow \langle\langle R,a\rangle\rangle; (x,z) \leftarrow \langle\langle R,b\rangle\rangle]$)

The view definition generated for $\langle\langle V\rangle\rangle$ would be

$$[(y,z) \mid (x,y,z) \leftarrow [(x,y,z) \mid (x,y) \leftarrow \langle\langle R,a\rangle\rangle; (x,z) \leftarrow \langle\langle R,b\rangle\rangle]]$$

and the generator $(x,y,z) \leftarrow$ in the outer comprehension can be fused with the head

expression of the inner comprehension, giving:

$$[(y,z) \mid (x,y) \leftarrow \langle\langle R,a\rangle\rangle; (x,z) \leftarrow \langle\langle R,b\rangle\rangle]$$

There are a range of other standard algebraic optimisations that could be performed on the view definitions *e.g.* pushing down selections and projections. However, these kinds of optimisations will also be applied later, when a specific global query is reformulated by substituting into it the view definitions. Further optimisations and rewrites will be applied at this stage *e.g.* to bring constructs from the same local schemas together into sub-queries which can be posed entirely on one local schema and it is these sub-queries (appropriately translated) that will be sent to local data sources for evaluation.

We finally note that, although IQL is list-based, if the ordering of elements within lists is ignored then its operators are faithful to the expected bag semantics. Moreover, use of the distinct operator can be used to obtain set semantics as needed. We refer the reader to [17, 6] for more details of IQL and for references to work on fold-based functional query languages and optimisation techniques for such languages.

5 Validating and Optimising Pathways

One important feature of the AutoMed approach is that once a set of schemas have been joined in a network of pathways, data and queries may be translated or migrated between any pair of schemas in the network. Such networks may be complex to analyse, so we need to support automated validation that a network is well-formed. We also need to support automated optimisation of the pathways between schemas, since they may contain redundant transformations.

To support such validation and optimisation of pathways, we have developed the **Transformation Manipulation Language** (**TML**) [20, 21], which represents each transformation in a form suited to analysis of the schema constructs that are created, deleted or are required to be present for the transformation to be correct. Our definitions below require two functions sc and rc. Given a query q on schema S containing n number of constructs, sc determines all schema constructs that must exist in S if the query is valid, rc determines all schema constructs in S referencing the constructs in q. For the IQL language constructs used in our earlier examples, sc and rc are defined as:

$$\begin{aligned}
sc(\langle\langle \mathsf{r}\rangle\rangle) &= \langle\langle \mathsf{r}\rangle\rangle \\
sc(\langle\langle \mathsf{r},\mathsf{a}\rangle\rangle) &= \{\langle\langle \mathsf{r}\rangle\rangle, \langle\langle \mathsf{r},\mathsf{a}\rangle\rangle\} \\
sc([q_1,\ldots,q_n]) &= sc(q_1) \cup \ldots \cup sc(q_n) \\
sc(q_1 \mathbin{++} q_2) &= sc(q_1) \cup sc(q_2) \\
sc([q \mid q_1,\ldots,q_n]) &= sc(q) \cup sc(q_1) \cup \ldots \cup sc(q_n) \\
rc(\langle\langle \mathsf{r}\rangle\rangle) &= \bigcup_{1\leq i\leq n} c_i (c_i \in S \wedge \langle\langle \mathsf{r}\rangle\rangle \in sc(c_i))
\end{aligned}$$

Note that as a shorthand, we will write the pair of queries q_l, q_u in extend or contract as just q, with the semantics in such cases that $sc(q_l, q_u) = sc(q_l) \cup sc(q_u)$. The TML formalises each transformation t_i of schema S_i into schema S_{i+1} as having four **conditions** $a_i^+, b_i^-, c_i^+, d_i^-$:

- The positive precondition a_i^+ is the set of constructs that t_i implies must be present in S_i. It comprises those constructs that are present in the query of the transformation (given by $sc(q)$) together with any constructs implied as being present by the construct c:

$t_i \in \{\mathsf{add}(c,q), \mathsf{extend}(c,q)\} \rightarrow a_i^+ = (sc(c) - c) \cup sc(q)$
$t_i \in \{\mathsf{delete}(c,q), \mathsf{contract}(c,q), \mathsf{id}(c,c')\} \rightarrow a_i^+ = sc(c) \cup sc(q)$
$t_i = \mathsf{rename}(c,c') \rightarrow a_i^+ = rc(c)$

- The negative precondition b_i^- is the set of constructs that t_i implies must not be present in S_i. It comprises those constructs which the transformation will add to the schema, and thus must not already be present:
  $t_i \in \{\mathsf{add}(c,q), \mathsf{extend}(c,q), \mathsf{id}(c',c)\} \rightarrow b_i^- = c$
  $t_i \in \{\mathsf{delete}(c,q), \mathsf{contract}(c,q)\} \rightarrow b_i^- = \emptyset$
  $t_i = \mathsf{rename}(c,c') \rightarrow b_i^- = \{c/c'\}rc(c)$
- The positive postcondition c_i^+ is the set of constructs that t_i implies must be present in S_{i+1}, and is derived in the same way as $\overline{a_i^+}$ (*i.e.* the positive precondition of the $\overline{t_i}$):
  $t_i \in \{\mathsf{add}(c,q), \mathsf{extend}(c,q), \mathsf{id}(c',c)\} \rightarrow c_i^+ = sc(c) \cup sc(q)$
  $t_i \in \{\mathsf{delete}(c,q), \mathsf{contract}(c,q)\} \rightarrow c_i^+ = (sc(c) - c) \cup sc(q)$
  $t_i = \mathsf{rename}(c,c') \rightarrow c_i^+ = \{c/c'\}rc(c)$
- The negative postcondition d_i^- is the set of constructs that t_i implies must not be present in S_{i+1}, and is derived in the same way as $\overline{b_i^-}$:
  $t_i \in \{\mathsf{delete}(c,q), \mathsf{contract}(c,q), \mathsf{id}(c,c')\} \rightarrow d_i^- = c,$
  $t_i \in \{\mathsf{add}(c,q), \mathsf{extend}(c,q)\} \rightarrow d_i^- = \emptyset$
  $t_i = \mathsf{rename}(c,c') \rightarrow d_i^- = rc(c)$

Below we show how the compounded transformation t_1 and the primitive transformation t_6 are represented in the TML.

$t_{1.1} : [\emptyset, \{\langle\langle \mathsf{student} \rangle\rangle\}, \{\langle\langle \mathsf{student} \rangle\rangle\}, \emptyset]$
$t_{1.2} : [\emptyset, \{\langle\langle \mathsf{student}, \mathsf{id} \rangle\rangle\}, \{\langle\langle \mathsf{student} \rangle\rangle, \langle\langle \mathsf{student}, \mathsf{id} \rangle\rangle\}, \emptyset]$
$t_{1.3} : [\emptyset, \{\langle\langle \mathsf{student}, \mathsf{id} \rangle\rangle\}, \{\langle\langle \mathsf{student} \rangle\rangle, \langle\langle \mathsf{student}, \mathsf{sex} \rangle\rangle\}, \emptyset]$
$t_{1.4} : [\emptyset, \{\langle\langle \mathsf{student}, \mathsf{id} \rangle\rangle\}, \{\langle\langle \mathsf{student} \rangle\rangle, \langle\langle \mathsf{student}, \mathsf{dname} \rangle\rangle\}, \emptyset]$
$t_6 : [\{\langle\langle \mathsf{staff} \rangle\rangle, \langle\langle \mathsf{staff}, \mathsf{dname} \rangle\rangle\}, \{\langle\langle \mathsf{dept} \rangle\rangle\}, \{\langle\langle \mathsf{dept} \rangle\rangle, \langle\langle \mathsf{staff} \rangle\rangle, \langle\langle \mathsf{staff}, \mathsf{dname} \rangle\rangle\}, \emptyset]$

5.1 Well-formed Transformation Pathways

A pathway T from schema S_m to S_n is said to be **well-formed** if for each transformation step $t_i : S_i \rightarrow S_{i+1}$ within it:

- The only difference between the schema constructs in S_{i+1} and S_i is those constructs specifically changed by transformation t_i, implying that $S_{i+1} = (S_i \cup c_i^+) - d_i^-$ and $S_i = (S_{i+1} \cup a_i^+) - b_i^-$
- The constructs required by t_i are in the schemas, implying that $a_i^+ \subseteq S_i$, $b_i^- \cap S_i = \emptyset$, $c_i^+ \subseteq S_{i+1}$ and $d_i^- \cap S_{i+1} = \emptyset$

The above definition leads to the recursive definition of a well-formed pathway, wf, given below. The first rule applies each transformation step in turn, and the second rule ensures that the schema that results from applying all the transformation steps is equal to the schema at the end of the pathway (equal both in terms of the schema constructs found in each schema and the extent of the schemas). Note that any implementation

may use these rules in two ways. Firstly, given a schema S_m representing a data source, and pathway P, a new data source schema S_n and its extent can be derived. Secondly, if S_n exists as a data source already, a check can be made to verify that P correctly derives its schema and extent from that of S_m.

$$wf(S_m, S_n, [t_m, t_{m+1}, \ldots, t_{n-1}]) \leftarrow a_m^+ \subseteq S_m \wedge b_m^- \cap S_m = \emptyset \wedge wf((S_m \cup c_m^+) - d_m^-, S_n, [t_{m+1}, \ldots, t_{n-1}])$$

$$wf(S_m, S_n, []) \leftarrow S_m = S_n \wedge Ext(S_m) = Ext(S_n)$$

5.2 Reordering of Transformations

Certain transformations may be performed in any order, whilst others must be performed in a specific order. For example, in $\mathsf{LS}_1 \rightarrow \mathsf{US}_1$, t_{11} must be performed before t_{12}, since the attribute $\langle\langle \mathsf{male}, \mathsf{id} \rangle\rangle$ must be deleted before the $\langle\langle \mathsf{male} \rangle\rangle$ relation is deleted. However the sub-pathway t_{11},t_{12} could be performed before or after the sub-pathway t_{13},t_{14} since it does not matter which of the $\langle\langle \mathsf{male} \rangle\rangle$ or $\langle\langle \mathsf{female} \rangle\rangle$ relations is deleted first.

In the TML, this intuition is expressed by stating that transformations may be swapped provided the pathway remains well-formed. This may be verified by inspecting the conditions of each transformation. In particular, a pair of transformations t_i,t_{i+1} may be reordered to t_{i+1},t_i provided:

1. t_i does not add a construct required by t_{i+1}, and $\overline{t_{i+1}}$ does not add a construct required by $\overline{t_i}$, *i.e.* $(c_i^+ - a_i^+) \cap a_{i+1}^+ = \emptyset$ and $(a_{i+1}^+ - c_{i+1}^+) \cap c_i^+ = \emptyset$
2. t_i does not delete a construct required not to be present by t_{i+1}, and $\overline{t_{i+1}}$ does not delete a construct required not to be present by $\overline{t_i}$, *i.e.* $d_i^+ \cap b_{i+1}^+ = \emptyset$
3. if t_i is preceded by t_{i-1}, the preconditions of t_{i+1} do not conflict with the postconditions of t_{i-1}, *i.e.* $c_{i-1}^+ \cap b_{i+1}^- = \emptyset$ and $d_{i-1}^- \cap a_{i+1}^+ = \emptyset$
4. if t_{i+1} is followd by t_{i+2}, the preconditions of t_{i+2} do not conflict with the postconditions of t_i, *i.e.* $c_i^+ \cap b_{i+2}^- = \emptyset$ and $d_i^- \cap a_{i+2}^+ = \emptyset$

We can now formalise the two examples given above from $\mathsf{LS}_1 \rightarrow \mathsf{US}_1$. For t_{11},t_{12}, (1) is broken, and hence they may not be swapped. The changing of t_{11},t_{12},t_{13},t_{14} to t_{13},t_{14},t_{11},t_{12} may be performed by iteratively swapping pairs of transformations. Considering first t_{12},t_{13}, we find neither rule is broken, and they may be reordered to t_{13},t_{12}. Then t_{12},t_{14} breaks neither rule, and may be reordered to t_{14},t_{12}. This leaves a sub-pathway t_{11},t_{13},t_{14},t_{12}, and a similar argument allows t_{11} swap with t_{13} and then t_{14}, to give the sub-pathway t_{13},t_{14},t_{11},t_{12}.

5.3 Redundant anad Partially Redundant Transformations

Two transformations t_x and t_y in a well-formed pathway T are **redundant** if T may be reordered such that t_x and t_y become consecutive within it, and T remains well-formed if they are then removed. Such redundant transformations will occur if a source schema evolves to model information in the same way as the global schema when previously it modelled the information in a different way. For example, suppose LS_1 is evolved by transformations t_{49},t_{50},t_{51},t_{52},t_{53}, textually identical to transformations t_{10},t_{11},t_{12},t_{13},t_{14}, to model the gender of staff as a single sex attribute in a new version

of the schema LS'_1. By reversing these transformation steps we can derive the pathway from the new to the old schema $\mathsf{LS}'_1 \rightarrow \mathsf{LS}_1$:

Example 5 Pathway $\mathsf{LS}'_1 \rightarrow \mathsf{LS}_1$
$\overline{t_{53}}$ addRel(⟨⟨female⟩⟩, [x | (x, 'F') ← ⟨⟨staff, sex⟩⟩])
$\overline{t_{52}}$ addAtt(⟨⟨female, id⟩⟩, [(x, x) | x ← ⟨⟨female⟩⟩])
$\overline{t_{51}}$ addRel(⟨⟨male⟩⟩, [x | (x, 'M') ← ⟨⟨staff, sex⟩⟩])
$\overline{t_{50}}$ addAtt(⟨⟨male, id⟩⟩, [(x, x) | x ← ⟨⟨male⟩⟩])
$\overline{t_{49}}$ deleteAtt(⟨⟨staff, sex⟩⟩, [(x, 'M') | x ← ⟨⟨male⟩⟩] ++ [(x, 'F') | x ← ⟨⟨female⟩⟩])

If we inspect the entire path $\mathsf{LS}'_1 \rightarrow \mathsf{US}_1$, consisting of $\mathsf{LS}'_1 \rightarrow \mathsf{LS}_1$ followed by $\mathsf{LS}_1 \rightarrow \mathsf{US}_1$, it may be reordered to contain the sub-pathway:
$\overline{t_{51}}$ addRel(⟨⟨male⟩⟩, [x | (x, 'M') ← ⟨⟨staff, sex⟩⟩])
$\overline{t_{50}}$ addAtt(⟨⟨male, id⟩⟩, [(x, x) | x ← ⟨⟨male⟩⟩])
$\overline{t_{49}}$ deleteAtt(⟨⟨staff, sex⟩⟩, [(x, 'M') | x ← ⟨⟨male⟩⟩] ++ [(x, 'F') | x ← ⟨⟨female⟩⟩])
t_{10} addAtt(⟨⟨staff, sex⟩⟩, [(x, 'M') | x ← ⟨⟨male⟩⟩] ++ [(x, 'F') | x ← ⟨⟨female⟩⟩])
t_{11} deleteAtt(⟨⟨male, id⟩⟩, [(x, x) | x ← ⟨⟨male⟩⟩])
t_{12} deleteRel(⟨⟨male⟩⟩, [x | (x, 'M') ← ⟨⟨staff, sex⟩⟩])

Clearly $\overline{t_{49}}$,t_{10} forms a redundant pair, because we are adding and deleting the same construct *with the same extent* since the query is the same. Once this has been performed $\overline{t_{50}}$,t_{11} may be removed for the same reason, and then $\overline{t_{51}}$,t_{12}. Once all other redundant pairs have been removed, $\mathsf{LS}'_1 \rightarrow \mathsf{US}_1$ would comprise of just t_1–t_9.

Using the TML, we can identify redundant transformations as satisfying:
$(a_x^+ = c_y^+) \wedge (b_x^- = d_y^-) \wedge (c_x^+ = a_y^+) \wedge (d_x^- = b_y^-) \wedge\ Ext(c_x^+ \oplus a_x^+) = Ext(c_y^+ \oplus a_y^+)$
where $(x \oplus y) = (x - y) \cup (y - x)$, and thus serves to find all the constructs being added or deleted by the pair of transformations. In practice, this rule means that any pair of transformations which add/extend and then delete/contract (in either order) the same construct are redundant, providing the query can be demonstrated to result in the same extent.

Two transformations t_x and t_y in a well-formed pathway T are **partially redundant** if T may be reordered to make t_x and t_y consecutive, and T remains well-formed if they are then replaced by a single transformation t_{xy}.

The pathway $\mathsf{LS}_1 \rightarrow \mathsf{LS}_2$ has a pair of such partially redundant transformations, since it can be reordered to obtain the sub-pathway:
t_7 addAtt(⟨⟨dept, dname⟩⟩, [(x, x) | x ← ⟨⟨dept⟩⟩])
$\overline{t_{18}}$ renameAtt(⟨⟨dept, dname⟩⟩, ⟨⟨dept, deptname⟩⟩)

This may be replaced by the new transformation given below, which leaves a fully optimised pathway $\mathsf{LS}_1 \rightarrow \mathsf{LS}_2$.
t_{54} addAtt(⟨⟨dept, deptname⟩⟩, [(x, x) | x ← ⟨⟨dept⟩⟩])

Using the TML, we can identify partially redundant transformations as satisfying the following rules, where $\oplus$ indicates the exclusive-or operator:
$((a_x^+ = c_y^+) \wedge a_x^+ \neq \emptyset \oplus (b_x^- = d_y^-) \wedge b_x^- \neq \emptyset \wedge d_x^- \cap b_y^- = \emptyset \wedge d_x^- \neq \emptyset \wedge b_y^- \neq \emptyset$

The simplifications for removing partially redundant and fully redundant transformations are summarised in the table below. The table shows what simplifications may be applied where a pair of transformations is found to operate on the same construct c. NWF denotes 'not well-founded' and [] denotes the removal of the pair. The table would remain correct if extend were to replace add, contract replace delete, and id

replace rename. Further details of redundant and partially redundant transformations may be found in [20, 21].

		t_y		
	add(c,q)	delete(c,q)	rename(c,c')
	add(c,q)	NWF	[ ]	add(c',q)
t_x	delete(c,q)	[ ]	NWF	NWF
	rename(c',c)	NWF	delete(c',q)	[ ]
	rename(c",c)	NWF	delete(c",q)	rename(c",c')

6 Concluding Remarks

In this paper we have described view generation and view optimisation in the AutoMed heterogeneous database integration framework. We have shown how the AutoMed schema pathways and views generated from them are amenable to considerable simplification, resulting in view definitions that look much like the views that would have been specified directly in a GAV, LAV or GLAV framework.

Since BAV integration is based on sequences of primitive schema transformations, it could be argued that data integration using it is more complex than with GAV, LAV or GLAV. However, the integration process can be greatly simplified by specifying well-known schema equivalences as higher-level composite transformations. We gave such an example, extendTable, in Section 2.1 above, and further examples are given in [15]. Moreover, we are working on techniques for semi-automatically generating transformation pathways to convert a source schema expressed in one modelling language into an equivalent target schema expressed in another modelling language, based on well known schema equivalences. We are also investigating schema matching techniques to automatically or semi-automatically integrate two specific schemas.

Finally, it should be noted that BAV is well-suited to peer-to-peer data integration (see [16]) since it lacks the directionality inherent in LAV, GAV and GLAV, all of which are tied to the concept of there being a global schema which may not always be the case in peer-to-peer environments.

References

1. M. Boyd, S. Kittivoravitkul, C. Lazanitis, P.J. McBrien, and N. Rizopoulos. AutoMed: A BAV data integration system for heterogeneous data sources. In *Proc. CAiSE2004*, 2004.
2. P. Buneman *et al*. Comprehension syntax. *SIGMOD Record*, 23(1):87–96, 1994.
3. D. Calvanese, E. Damagio, G. De Giacomo, M. Lenzerini, and R. Rosati. Semantic data integration in P2P systems. In *Proc. DBISP2P*, Berlin, Germany, 2003.
4. S.S. Chawathe *et al*. The TSIMMIS project: Integration of heterogeneous information sources. In *Proc. 10th Meeting of the Information Processing Society of Japan*, pages 7–18, October 1994.
5. M. Friedman, A. Levy, and T. Millstein. Navigational plans for data integration. In *Proc.16th National Conf. on AI*, pages 67–73. AAAI Press, 1999.
6. E. Jasper, A. Poulovassilis, and L. Zamboulis. Processing IQL Queries and Migrating Data in the AutoMed toolkit. Technical Report No. 20, AutoMed, 2003.

7. M. Lenzerini. Data integration: A theorectical perspective. In *Proc. PODS02*, pages 247–258, 2002.
8. A.Y. Levy, A.O. Mendelzon, Y. Sagiv, and D. Srivastava. Answering queries using views. In *Proc. PODS'95*, pages 95–104. ACM Press, May 1995.
9. A.Y. Levy, A. Rajamaran, and J.Ordille. Querying heterogeneous information sources using source description. In *Proc. VLDB'96*, pages 252–262, 1996.
10. J. Madhavan and A.Y. Halevy. Composing mappings among data sources. In *Proc. 29th Conference on VLDB*, pages 572–583, 2003.
11. I. Manolescu, D. Florescu, and D. Kossmann. Answering XML queries on heterogeneous data sources. In *Proc. VLDB'01*, pages 241–250, 2001.
12. P.J. McBrien and A. Poulovassilis. Automatic migration and wrapping of database applications — a schema transformation approach. In *Proc. ER'99, LNCS 1728*, pages 96–113, 1999.
13. P.J. McBrien and A. Poulovassilis. A uniform approach to inter-model transformations. In *Proc. CAiSE'99, LNCS 1626*, pages 333–348, 1999.
14. P.J. McBrien and A. Poulovassilis. Schema evolution in heterogeneous database architectures, a schema transformation approach. In *Proc. CAiSE'02, LNCS 2348*, pages 484–499, 2002.
15. P.J. McBrien and A. Poulovassilis. Data integration by bi-directional schema transformation rules. In *Proc. ICDE'03*, 2003.
16. P.J. McBrien and A. Poulovassilis. Defining peer-to-peer data integration using both as view rules. In *Proc. DBISP2P*, Berlin, Germany, 2003.
17. A. Poulovassilis. The AutoMed Intermediate Query Language. Technical report, AutoMed Project, 2001.
18. M.T. Roth and P. Schwarz. Don't scrap it, wrap it! A wrapper architecture for data sources. In *Proc. VLDB'97*, pages 266–275, Athens, Greece, 1997.
19. M. Templeton, H.Henley, E.Maros, and D.J. Van Buer. InterViso: Dealing with the complexity of federated database access. *VLDB Journal*, 4(2):287–317, 1995.
20. N. Tong. Database schema transformation optimisation techniques for the AutoMed system. Technical report, AutoMed Project, 2002.
21. N. Tong. Database schema transformation optimisation techniques for the AutoMed system. In *Proc. BNCOD'03*, volume 2712 of *LNCS*, pages 157–171. Springer, 2003.

SCIENTIFIC PAPERS UNIVERSITY OF LATVIA. 2004. VOLUME 672, Datbases and Information Systems
31–46 P.

A Heuristic Approach to Horizontal Fragmentation in Object Oriented Databases

Hui Ma, Klaus-Dieter Schewe

Massey University, Department of Information Systems &
Information Science Research Centre
Private Bag 11 222, Palmerston North, New Zealand
[h.ma|k.d.schewe]@massey.ac.nz

Abstract. Distribution design for databases involves fragmentation, allocation and replication. This paper addresses fragmentation in the context of a structurally rich object oriented data model. Horizontal and vertical fragmentation operations are extended from the relational data model to the object oriented data model, and a third fragmentation operation called splitting is introduced. The goodness of a distribution design should be assessed by the performance of a system. For this we present a query processing cost model to evaluate the performance of a system. The core of the paper is a heuristic approach for horizontal fragmentation, which uses the cost model and is targeted at globally minimising these costs.

Keywords. fragmentation, distribution design, cost model, heuristic procedure

1 Introduction

It is commonly accepted that the design of distributed databases extends the design of centralised databases by three additional activities: fragmentation, allocation and replication [4, 13]. Fragmentation replaces relation schemata or classes in a given database schema by several new relation schemata or classes without losing or adding information. Allocation associates with each fragment a node in the computer network. Thus, fragmentation and allocation together replace a global database schema by local database schemata for each node of the network. In addition, replication allows a fragment to be stored not just on one node, but on several ones, which impacts on the way queries and updates using replicated fragments have to be realised.

The main objective of distribution design is to improve the overall performance and reliability of a distributed database system. Improved performance can be achieved by avoiding or at least reducing unnecessary data transport between the nodes of the network.

Fragmentation and allocation have mainly been investigated in the context of the relational data model. Horizontal fragmentation [8, 14, 19] exploits relations being sets of tuples. These sets are partitioned using selection predicates.

Depending on whether these selection predicates are based on the relation schema at hand or on another relation we distinguish between primary and derived horizontal fragmentation. Vertical fragmentation [12, 13] starts from relation schemata as sets of attributes. Fragments are defined by projections onto subsets and the original relation is recovered by a natural join. This join must be lossless, which is usually achieved by letting a selected minimal be part of all fragment relation schemata. Alternatively, an additional virtual attribute can be used. Allocation [2] assigns the fragments to the nodes in a network. More generally, we may extend allocation to associate nodes also to all intermediate query results, but to our knowledge this generalised view has not yet been approached. In most work in the literature fragmentation and allocation are studied separately, but in some papers these activities are treated jointly [18].

Search at the DBLP digital bibliography service (see `http://dblp.uni-trier.de`) shows that distribution design has been a hot research topic in the early years of last decade, but the interest in the topic has decreased despite the fact that the problems still need further investigation. Especially, with the advent of coupling databases and web-based systems using XML, we obviously create distributed databases, but the way we do it is more ad-hoc than planned and systematically. This current trend also indicates the need for understanding database distribution on the basis of more sophisticated data models than the relational one, in particular object oriented data models and similarly semi-structured data and XML. However, most of the research on distribution design is still based on the relational model, and very little attention is paid to other data models, especially object oriented models.

The work in [3, 6, 9, 16] generalises horizontal fragmentation to the object oriented case. The work in [5, 7, 11, 16] does the analogue for vertical fragmentation. The work in [10] contains first steps to carry fragmentation over to XML. In a nutshell, common horizontal fragmentation techniques start from simple selection predicates found in the most frequent queries and their usage in these queries. The fragments are defined by clustering these simple predicates using either graphical techniques [14] or variants of the Bond Energy Algorithm (BEA) [19], which has originally been used in the context of vertical fragmentation. Vertical fragmentation is based on the clustering of attributes using the affinity between them. Again, approaches are graph-based [12] or use BEA [13]. Most of the work on horizontal and vertical fragmentation follows these basic ideas.

The existing approaches to fragmentation in the context of object oriented databases suffers from three major drawbacks:

- The assumed data model is usually extremely simplified in comparison to more advanced models such as the OODM [17] or IQL [1]. In particular, they miss out on deeply nested structures and consequently on splitting as a primitive for fragmentation [16].

- Despite the fact that distribution design is largely concerned with performance optimisation, little reference is made to the actual query (and transaction) computing costs. Intermediate results of the queries, which have to be stored somewhere in the network and transported to other nodes should be taken into account [9].
- Approaches rely on dependencies between simple queries, though these can hardly be determined. In the end the major use of these dependencies is to determine the satisfiability of a conjunction of simple predicates which is undecidable, and if satisfiable to simplify them.

Our work here is a first attempt to overcome these drawbacks. The remainder of the paper is organised as follows. The fundamentals of the object oriented data model will be reviewed in Section 2 as the preliminaries of the following discussion. In Section 3 we will review fragmentation techniques for the OODM in [16] which leads us to distinguish not only horizontal and vertical fragmentation, but also splitting. Then in Section 4 we present a general query processing cost model and use it as the basis for a new heuristic method for cost optimisation based on reducing the number of relevant simple predicates and thus fragments. The assumption underlying the heuristics is that building fragments with respect to the most frequent simple predicates will first reduce costs, but using too many of them will increase costs again. The approach will determine a suitable cut-off point. Finally Section 5 presents a brief conclusion.

It has been argued that object oriented databases are out of date and after all have not fulfilled the expectations. Whether this is true or not, the object oriented model we use captures the gist of XML, and thus the approach to fragmentation can be carried over. In [16] it has already been shown how to carry over the fragmentation operations to semi-structured data. In [10] horizontal fragmentation has been approached for XML documents.

2 Fundamentals of the OODM

The OODM in [17] is based on an underlying type system. For our purposes here it is sufficient to consider a type system defined as follows:

$$t = b \mid x \mid (a_1 : t_1, \ldots, a_n : t_n) \mid \{t\} \mid (a_1 : t_1) \cup \cdots \cup (a_n : t_n).$$

Here b represents base types, $(\cdot)$ a record type constructor, $\{\cdot\}$ a set type constructor, and $\cup$ a union type constructor. x is used for type variables, here only for the purpose of defining classes. In order to have object identifiers, we require that among the base types there is at least one type ID, the values of which are the oid-s.

2.1 Schemata and Databases

Now consider a type with variables $x_1, \ldots, x_n$, but with no occurrence of ID. Replacing all these variables x_i by pairs $r_i : C_i$ with some mutually distinct

labels r_i (called *references*) and names C_i (called *class names*) results in a *structure expression*.

A *class* consists of a class name C, a structure expression exp_C, and a set $\{C_1, \ldots, C_n\}$ of class names (called the *superclasses*). If we replace all $r_i : C_i$ in exp_C by the base type *ID*, the resulting parameter-less type T_C is called the *representation type* of the class C.

A *schema* is a finite set of classes that is closed in the sense that all class names appearing in a structure expression or as a superclass must be names of classes defined in the schema.

Example 2.1. Choose the following schema as an example:

`Class` REGION `Struct` (name: *STRING*, central : h: CITY,
cities : { c : CITY })

`Class` CITY `Struct` (name: *STRING*, in : r: REGION, population: *NAT*)

A *database db* for a schema $\mathcal{S}$ assigns to each class $C \in \mathcal{S}$ a finite set $db(C)$ of pairs (i, v), where i is an oid, i.e. a value of type *ID*, and v is a value of type t_C, such that certain conditions are satisfied. Informally, these conditions are the uniqueness of identifiers, the inclusion of the set of oid-s in a class in the set of oid-s for each of its subclasses (inclusion integrity), the appearance of each oid occurring in a value v as an oid i in the referenced class (referential integrity), and (weak) value-identifiability. For a formal treatment of these conditions we refer to [17].

2.2 A Simple Query Algebra

In order to define a query algebra we could follow the unified and generalised approach in [15], but for our purposes here we can use a simpler algebra. Other operations including nesting and unnesting can be defined explicitly in this section.

The operations of the algebra must allow us to refer to elements of classes and their components. For this we use path expressions. Let C be a class with structure expression exp_C and representation type T_C. *Path expressions* for class C (or exp_C) have one of the following formats:

- *ident* of type ID, or *value* of type T_C,
- If exp_C uses a record type, i.e. $exp_C = (a_1 : exp_1, \ldots, a_n : exp_n)$, then we get path expressions
  - $value.a_i$ of type T_i which is the representation type for exp_i and $1 \leq i \leq n$,
  - $value!a_i$ of type T_D if a_i is a reference to class D, i.e. $a_i : exp_i = a_i : D$,
  - $value.a_i.path_i$ where $path_i$ is a path expression for exp_i,
  - $value!a_i.path_i$ where a_i is a reference to class D, i.e. $a_i : exp_i = a_i : D$ and $path_i$ is a path expression for exp_D.

- If exp_C use a union type, i.e. $exp_C = (a_1 : exp_1) \cup \cdots \cup (a_n : exp_n)$, then we get path expressions
  - $value.a_i$ of type T_i which is the representation type for exp_i,
  - $value.a_i.path_i$ where $path_i$ is a path expression for exp_i.
- If exp_C uses a set type, i.e. $exp_C = \{exp\}$, then we obtain path expressions
  - $value$ of type $T_C = \{exp\}$,
  - $value.path$ where $path$ is a path expression corresponding to exp.
- If exp_C uses only a reference , i.e. $exp_C = r : D$, then we obtain a path expression $value!r.path$ where $path$ is path expression for the class D.

In the OODM each query should result in a set of pairs (i, v), where i is an identifier and v is a value of some proper type. The value v may contain identifiers, which must appear in the database or in the query result.

A *query* Q on $\mathcal{S}$ consists of a structure expression exp_Q called answer schema (with all references pointing to classes in $\mathcal{S}$) and an algebra expression q, i.e. a query Q is defined in the form $Q = (exp_Q, q)$. The operators in the query algebra take one or two class instances as arguments and return another class instance as a result.

Let $\mathcal{S}$ be a database schema, db be a database instance over $\mathcal{S}$, $id(db)$ be the set of all identifiers appearing in db, $db(Q)$ be the resulting class instance of evaluating query algebra $Q = (exp_Q, q)$. The query algebra operations are defined as follows:

- $q = C$ with C is a class name appearing in $\mathcal{S}$, resulting in $db(Q) = \{(id_w, v) \mid id_w : ID, id_w \notin id(db).\exists id : ID.(id, v) \in db(C)\}$;
- $q = (v : T)$ with a type T and a value v of type T and $Q = (T, q)$, resulting in $db(Q) = \{(id_w, v) \mid id_w : ID \wedge id_w \notin id(db)\}$;
- $q' = \sigma_\varphi(Q)$ (selection) with a selection formulae φ that is defined either as $path_1 = path_2$ or as $path = v$, resulting in $db(Q') = \{(id_w, v) \mid id_w : ID \wedge id_w \notin id(db).\exists id : ID.(id, v) \in db(Q) \wedge \varphi(v) = true\}$;
- $q' = \rho_{a_1 \rightarrowtail b_1, \ldots, a_n \rightarrowtail b_n}(Q)$ (renaming) resulting in $db(Q') = \{(id_w, v) \mid id_w : ID \wedge id_w \notin id(db).\exists id : ID.(id, v) \in db(Q)\}$ with b_i as a new name for attribute a_i;
- $q' = \pi_{Q'}(Q)$ (generalised projection) with a new structure expression $exp_{Q'}$ that is a super structure expression of exp_Q, and resulting in $db(Q') = \{(id_w, v') \mid id_w : ID \wedge id_w \notin id(db).\exists id : ID.(id, v) \in db(Q).v' = \pi^Q_{Q'}(v)\}$;
- $q' = q_1 \bowtie_{exp} q_2$ (generalised join) with a common super structure expression, i.e. $exp_{Q_i} \leq exp$ for $i = 1, 2$, resulting in

$$db(Q_1 \bowtie_{exp} Q_2) = \{(id_w, v) \mid id_w : ID \wedge id_w \notin id(db).\exists id : ID.$$
$$(id_1, v_1) \in db(Q_1).(id_2, v_2) \in db(Q_2).\pi^{Q_1}_{exp}(v_1) = \pi^{Q_2}_{exp}(v_2) \wedge$$
$$\pi^{Q_1 \bowtie Q_2}_{Q_1}(v) = v_1 \wedge \pi^{Q_1 \bowtie Q_2}_{Q_2}(v) = v_2\};$$

$$db(Q') = \{(id_w, w) \mid id_w : ID \wedge id_w \notin id(db).\exists id : ID.(id_1, v_1) \in db(Q).$$
$$\pi^Q_{Q-exp}(w) = \pi^Q_{Q-exp}(v_1) \wedge \pi^Q_{exp}(w) = \{\pi^Q_{exp}(v_2) \mid (id_2, v_2)$$
$$\in db(Q) \wedge \pi^Q_{Q-exp}(v_1) = \pi^Q_{Q-exp}(v_2)\}\}$$

with $exp_{Q'} = (exp_Q - exp) \cup \{exp\}$.

- unnest $q' = \mu_{exp}(Q)$ (unnest operation) with exp as a set attribute of Q resulting in

$$db(Q') = \{(id_w, w) \mid id_w : ID \wedge id_w \notin id(db).\exists id : ID.(id, v) \in db(Q).$$
$$\pi^Q_{Q-\{exp\}}(w) = \pi^Q_{Q-\{exp\}}(v) \wedge \pi^Q_{exp}(w) \in \pi^Q_{exp}(v)\}$$

with $exp_{Q'} = (exp_Q - \{exp\}) \cup exp$.

- the usual set operations $\cup$ (union), $-$ (difference), and $\cap$ (intersection).

Queries in this algebra can be rewritten in the usual way as labelled *query trees.*

2.3 Horizontal Fragmentation

In [16] three operations for fragmentation of object oriented databases have been discussed: splitting, horizontal fragmentation and vertical fragmentation. In this article we only refer to horizontal fragmentation, which we briefly review here.

According to the definition of databases for an OODM schema, each class will be associated with a set of pairs. Hence, we have the trivial generalisation of relational horizontal fragmentation.

For this let C be some class. Take boolean valued function φ_i such that for each database db we obtain $db(C) = \bigcup_{i=1}^{n} \sigma_{\varphi_i}(db(C))$ with disjoint sets $\sigma_{\varphi_i}(db(C))$.

We then replace C in the schema by n new classes C_i, all with $exp_{C_i} = exp_C$. However, as there may be classes D referencing C, i.e. $r : C$ occurs within exp_D, we have to replace this reference as well. This can be done by replacing $r : C$ in exp_D by $(a_1 : r_1 : C_1, \ldots, a_n : r_n : C_n)$ with new pairwise distinct reference names $r_1, \ldots, r_n$.

Example 2.2. Take the schema from Example 2.1 and fragment the class CITY using $\varphi_1 \equiv$ name $\leq$ 'Paraparaumu' and $\varphi_2 \equiv$ name $>$ 'Paraparaumu'.

Then the new schema will be

`Class` REGION `Struct` (name: *STRING*, central : (ltp : h_1: CITY_1) $\cup$ (gtp : h_2: CITY_2), cities : { (ltp : c_1: CITY_1) $\cup$ (gtp : c_2: CITY_2)})

`Class` CITY_1 `Struct` (name: *STRING*, in : r: REGION, population: *NAT*)

`Class` CITY_2 `Struct` (name: *STRING*, in : r: REGION, population: *NAT*)

3 A Heuristic Method for Horizontal Fragmentation

We now present a heuristic approach to horizontal fragmentation in the context of the OODM. The major objective of the approach is to base the fragmentation decision on the efficiency of the most frequent queries. Thus, we start analysing the selection predicates used in these queries. In the general literature [4, 13] the recommended rule of thumb is to consider only the 20% most frequent queries, as these are supposed to account for about 80% of the data access.

3.1 Normal Predicates

Let $Q_1, \ldots, Q_k$ be the queries we are interested in. According to our introduction of the query algebra we may assume to be given the query tree for each of them. Furthermore, we may tacitly assume that these query trees are 'optimised' by applying appropriate query optimisation techniques. Then the leaves of these trees correspond to basic queries in the form C for a class name C, and most predecessors of leaves will correspond to a selection query $\sigma_\varphi(C)$. The selection predicates φ in these queries give rise to simple predicates.

In general, a *simple predicate* for a class C has the form *path* θ v with a path expression *path* on C, a value v of the corresponding type, and a comparison operator Θ, which can be one of $=, \neq, \leq, <, >, \geq, \subseteq, \supseteq, \not\subseteq, \not\supseteq, \in, \notin, \ni$, and $\not\ni$.

Example 3.1. Choose the schema of class REGION from example 2.1:

`Class` REGION `Struct` (name: *STRING*, central : h: CITY,
cities : { c : CITY }).

There is a simple predicate defined on it for searching which region the city 'Paraparaumu' is in:

$$\varphi \equiv value!cities.name \ni \text{'Paraparaumu'}$$

Now let $\Phi = \{\varphi_1, \ldots, \varphi_m\}$ denote a set of simple predicates on a class C. Then the set of *normal predicates* $\mathfrak{N} = \{\mathcal{N}_1, \ldots, \mathcal{N}_n\}$ on class C is the set of all satisfiable predicates of the form $\mathcal{N}_j \equiv \varphi_1^* \wedge \cdots \wedge \varphi_m^*$, where φ_i^* is either φ_i or $\neg\varphi_i$.

Example 3.2. Take the class schema CITY from example 2.1:

`Class` CITY `Struct` (name: *STRING*, in : r: REGION, population: *NAT*).

A normal predicate can be defined as:

$$\mathcal{N}_j \equiv \underbrace{value.name \leq \text{'Marton'}}_{\varphi_1} \wedge \underbrace{value.population > 50000}_{\varphi_2}$$

Same as for the RDM [13] the normal predicates do not exhaust all possible selection formulae on a class C. However, as the OODM allows cyclic references to be used on a schema, there are infinitely many such selection formulae. Even

more, there are path expressions of arbitrary length, so we have no chance to capture the complete variety of selection. On the other hand, we want to define only finitely many fragments. So we do not lose much by restricting ourself to normal predicates.

Of course, the normal predicates derived from the simple predicates on class C can be used to define the horizontal fragments of class C. The idea of our heuristic procedure for horizontal fragmentation is to reduce the number of simple predicates taken into account and thus to reduce the number of horizontal fragments, if this reduces the query processing costs.

3.2 Size Calculation

Crucial to the query costs are the sizes of class instances. So let us first look at them. The calculation of the size

for classes is more complicated in the OODM than the calculation of size for relations in the relational model. In the relational model, we only use the record type constructor. Then the size of a relation can be calculated based on the size of a tuple. In the object oriented model, however, other type constructors such as set and maybe some other bulk type constructors are involved. The size calculation for classes based on the record type constructor only will occur as a special case in the OODM.

Let us first consider the simplest situation: a class C with a representation type $T_C = b$, with b indicating a base type. Let n denote the number of objects in a class instance $db(C)$ over T_C, l_v the average space (in bits) for a value of type b, and l_{id} the size of an identifier, i.e. a value of type ID. Then the average size of $db(C)$ is $s_C = n \cdot (l_v + l_{id})$.

Note that this basic case already includes references, which leads to the type $b = ID$ and $l'_v = l_{id}$.

Now let C be a class defined by a record type constructor, i.e. we have $T_C = (a_1 : T_1, \ldots, a_n : T_n)$. The calculation of the size of a class is very similar to that in the relational model. Let n denote the number of objects in a class instance $db(C)$ over T_C, $l_0 = l_{id}$ and l_{id} the size of an identifier, and l_j the average space (in bits) for field a_j of T_C. Then the average size of $db(C)$ over exp_C is $s_C = n \cdot \sum_{j=0}^{k} l_j$.

If there is a class C with a structure expression $T_C = \{exp\}$, the calculation of the size of each class instance is different from the first two cases. Let n denote the number of objects in a class instance $db(C)$ over T_C, l_{id} denote the size of an identifier, m the number of values of an object o in a class instance $db(C)$ over T_C, and l_v the average space (in bits) for a value over exp of an object o in a class instance. Then the average size of a object o over T_C is $l_{id} + m \cdot l_v$ and the average size of class over T_C is $s_C = n \cdot (l_{id} + m \cdot l_v)$.

However, the cost model is still not realistic. Even if we accept the inaccuracy resulting from using only mean values, it is hardly the case that a complex

value will be stored as a simple unit, especially if sets are involved. It is more likely to use data structures known from the network data model for this.

If T_C involves a set type constructor, i.e. $T_C = (a_1 : T_1, \ldots, a_i : \{T_i\}, \ldots, a_n : T_n)$, we would store values of type T_i separately as a linked list and just include a pointer to the first element of this list as well as a pointer back from the last element of the list.

This means that instead of l_v we have to consider $l_{id} + l_v$ assuming the l_{id} is the size of a pointer. Then the whole set value needs the space $l_{id} + m \cdot (l_{id} + l_v)$, where m is again the average number of elements in the set. Note that the size of a set is $m \cdot l_v + (m+1) \cdot l_{id}$. It means that we need $(m+1) \cdot l_{id}$ more space to store the values of a set if we treat it as being stored in a linked list with a pointer to an object o of a class C where m is the number of objects in the set.

3.3 A Query Processing Cost Model

The calculation of sizes of class instances applies also to the intermediate results of all queries. These intermediate results correspond to the non-leaf nodes of the query trees. In addition, these nodes correspond to a particular operation of the query algebra.

- The size of a selection node σ_φ is $p \cdot s$, where s is the size of the successor node and $100p$ is the average percentage of objects in the successor satisfying φ.
- The size of a projection node π_{exp} is $(1 - c_i) \cdot s \cdot \dfrac{l_f}{l_o}$ where $l_f(l_o)$ is the average size of an object over $exp(exp_C)$. s is the size assigned to the successor and c_i is the probability that two classes coincide on exp.
- For a join node the assigned size is $\dfrac{s_1}{l_1} \cdot p \cdot \dfrac{s_2}{l_2}(l_1 + l_2 - l)$, where s_i are the sizes of the successors, l_i are the corresponding object sizes, l is the size of a tuple over the common attributes and p is the matching probability.
- For a union node the size is $s_1 + s_2 - p \cdot s_1$ with the probability p for an object of C_1 to coincide with an object of C_2.
- For a difference node the assigned size is $s_1 \cdot (1 - p)$ with the probability p for an object of C_1 to coincide with an object of C_2.
- For a renaming node the assigned size is exactly the size s assigned to the successor.
- For a nest node the size is $s + n \cdot (m+1) \cdot l_{id} - n \cdot (m-1) l_{un}$ with l_{un} the length of the unnested attributes, m the average number of elements in the nested set, n the number of objects in a class instance $db(C)$ over T_C.
- For a unnest node the size is $s - n \cdot (m+1) \cdot l_{id} + n \cdot (m-1) l_{un}$ with again l_{un} the length of the unnested attributes, m the average number of elements in the nested set, n the number of objects in a class instance $db(C)$ over T_C.

Fragmentation of a class C results in a set of fragments $\{f_1, \ldots, f_n\}$ of average sizes $s_1, \ldots, s_n$, if the network has nodes $N_1, \ldots, N_k$ we have to allocate these fragments to the nodes, which gives rise to a mapping $\lambda : \{1, \ldots, n\} \to$

$\{1, \ldots, k\}$, which we call a *location assignment*. However, the fragments only appear on the leaves of query trees. More generally, we must associate a node $\lambda(v)$ with each node v in each relevant query tree. $\lambda(v)$ indicates the node in the network, at which the intermediate query result corresponding to v will be stored.

Given a location assignment λ we can compute the total costs of query processing. Let the set of queries be $Q^m = \{Q_1, \ldots, Q_m\}$. Query costs are composed of two parts: *storage costs* and *transportation costs*: $costs_\lambda(Q_j) = stor_\lambda(Q_j) + trans_\lambda(Q_j)$. The storage costs give a measure for retrieving the data back from secondary storage, which is mainly determined by the size of the data. The transportation costs provide a measure for transporting between two nodes of the network.

The storage costs of a query Q_j depend on the size of the involved classes or fragments, respectively, and on the assigned locations which decide the storage cost factors. It can be expressed as $stor_\lambda(Q_j) = \sum_h s(h) \cdot d_{\lambda(h)}$. where h ranges over the nodes of the query tree for Q_j, $s(h)$ are the sizes of the involved class instances, and d_i indicates the storage cost factor for node N_i $(i = 1, \ldots, k)$.

The transportation costs of query Q_j depend on the sizes of the involved classes or fragments of classes and on the assigned locations which decide the transport cost factor between every pair of sites. It can be expressed by $trans_\lambda(Q_j) = \sum_h \sum_{h'} c_{\lambda(h')\lambda(h)} \cdot s(h')$. Again the sum ranges over the nodes h of the query tree for Q_j and, h' runs over the predecessors of h in the query tree, and c_{ij} is a transportation cost factor for data transport from node N_i to node N_j $(i, j \in \{1, \ldots, k\})$.

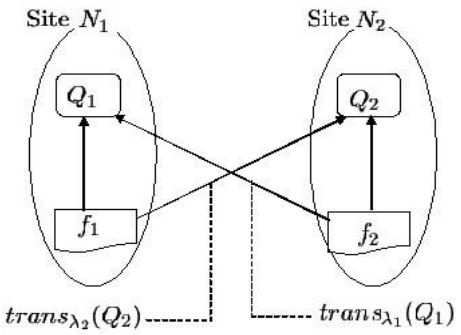


Fig. 1. Allocation of Fragments to Two Sites

For each query Q_j we get a value for its frequency $freq_j$. The total costs of all the queries in Q^m are the sum of the costs of each query multiplied by its frequency. It can be expressed by $cost_\lambda = \sum_{j=1}^{m} cost_\lambda(Q_j) \cdot freq_j$.

3.4 The Fragmentation Procedure

Fragmentation and allocation are generally considered as two isolated problems in [13] and [4]. After fragmentation, the fragments are allocated to reside at one node in a distributed management system (not considering replication at this stage in this paper). There is no cost model involved for the fragmentation design in [13]. But we argue that the values of the costs of queries after fragmentation will affect the decision on whether we need to perform fragmentation or not. A cost model should be used to evaluate different fragmentatin solutions. Let us look at the following example to see whether fragmentation of a class and allocating result fragments to different sites will achieve better performance.

Example 3.3. Consider a class being fragmented into two fragments f_1, f_2, and two queries Q_1, Q_2 executing at two different sites N_1, N_2 to access these two fragments remotely. The frequencies of Q_1 and Q_2 are $freq_1$ and $freq_2$, respectively. This design is shown in Figure 1.

If the sizes of f_1 and f_2 are s_1 and s_2, respectively, ignoring storing costs for all the fragments, we have total query costs of:

$$\begin{aligned} cost_{\lambda_1} &= trans_{\lambda_1}(Q_1) + trans_{\lambda_1}(Q_2) \\ &= s_2 \cdot freq_1 \cdot c_{21} + s_1 \cdot freq_2 \cdot c_{12} \end{aligned}$$

with c_{12} and c_{21} as transportation cost factors. Generally, c_{12} should be equal to c_{21}.

If query Q_1 is executed more frequently, say $freq_1 > freq_2$, we do not fragment the class and put the whole class at site N_1, the site that Q_1 is executed. This design is shown in Figure 2.

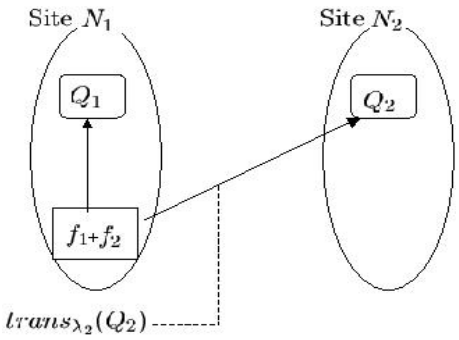


Fig. 2. Allocation of Class to One Site

The total query costs for the second design are:

$$cost_{\lambda_2} = trans_{\lambda_2}(Q_2) = (s_1 + s_2) \cdot freq_2 \cdot c_{12}$$

Comparing costs $cost_{\lambda_1}$ and $cost_{\lambda_2}$ we get $cost_{\lambda_1} > cost_{\lambda_2}$.

It can be concluded that fragmentation of classes does not always minimize the query cost. In general, the distribution could be called optimal, if we find a location assignment λ and a fragmentation solution such that the resulting total query costs are minimal. As this problem is computably intractable, we suggest to use a heuristic approach instead.

For a given database schema $\mathcal{S} = \{C_1, \ldots, C_i, \ldots C_n\}$, there is a set of queries $\{Q_1, \ldots, Q_m\}$ that access the database most frequently or that are used by the most critical transactions. Then the heuristic procedure of horizontal fragmentation includes the following steps:

1. Identify the set of most frequent queries accessing class C_i and rewrite them using the query algebra.
2. Identify the sites at which the queries will be issued. Treat queries that are started at several sites as several different queries. Estimate for each query the frequency of its execution.
3. Sort queries by their frequencies. Then we get a list of queries $[Q_1, \ldots, Q_j, \ldots, Q_m]$ with the corresponding list of values of frequencies $F^m = [freq_1, \ldots, freq_m]$ for Q^m, i.e. we always get $freq_{k-1} \geq freq_k$.
4. Determine optimized query trees for all the queries. Extract simple predicates from these trees. We get a set Φ of simple predicates φ.
5. Construct a usage matrix based on the simple predicates obtained in the previous steps, i.e. determine which simple predicates are used by which queries.
6. From the matrix obtained in the previous step we can get a list Φ_i^b of sorted simple predicates $\Phi_i^b = [\varphi_{i1}, \ldots \varphi_{ib}]$ where we always get $freq_{j-1} \geq freq_j$. The number of simple predicates is b. We get a list $X = [0, 1, \ldots, x_1, \ldots, x_2, \ldots, b]$ of indices for the simple predicates.
7. Perform the following steps iteratively to find a reasonable number of simple predicates for horizontal fragmentation and fragment the given class C_i: Choose four numbers a, b, x_1, x_2 from X with $a < x_1 < x_2 < b$, including initially the two bounds, i.e. start with $0, x_1, x_2, b$ with $0 < x_1 < x_2 < b$. For each number $x \in \{a, b, x_1, x_2\}$ we calculate the corresponding query costs by the following procedures:
   (a) Choose first x simple predicates in the list Φ_i^b, i.e. $\Phi_i^x = \{\varphi_{i1}, \ldots \varphi_{ix}\}$ and build the corresponding set of normal predicates $\mathcal{N}^x$.
   (b) Fragment the class C_i according to the set $\mathcal{N}^x$ of normal predicates obtained in the previous step. Let the fragments be $F_1^x, \ldots, F_r^x$.
   (c) Calculate the frequencies $freq_k^j$ of access to the fragments F_k from site j, and use this to determine fragment allocation to sites. Put the fragments to the nodes that access them most frequently.
   (d) Calculate the total query costs $cost_x$ with the cost model introduced above.

We get four values for the query costs $cost_a, cost_{x_1}, cost_{x_2}, cost_b$. Let min denote the minimal value among these four values. Comparing these query costs we might have the following four situations:

- If $min = cost_a$ then set $b = x_1$ and choose two new values for x_1, x_2 satisfying $a < x_1 < x_2 < b$. If we can find such two numbers then continue the procedure.
  If we can only get one number x from the list X, calculate the query cost for it. If $cost_x < cost_a$, then x should be the number of simple predicates that we should use to fragment the class C_i. Thus, let $y := x$.
  If $cost_x > cost_a$ or there is no number left between the new a and b, then a is the number of simple predicates that we will use for fragmentation. Thus, let $y := a$.
- If $min = cost_{x_1}$, then set $b = x_2$ and choose a new number between a and b so that we get new numbers for x_1, x_2. Continue with the procedure.
  If we can not find a new number between new a and b, then x_1 is the number of simple predicates that we should choose for fragmentation, thus $y := x_1$.
- If $min = cost_{x_2}$, then set $a = x_1$. Choose a new number between a and b so that we get new values for x_1, x_2 and continue the procedure with the new $a < x_1 < x_2 < b$. If we do not find such a new number, x_2 will be the number of simple predicates that we need for fragmenting the class C_i, i.e. take $y := x_2$.
- If $min = cost_b$, then set $a = x_2$ and find two new numbers between a and b satisfying the condition $a < x_1 < x_2 < b$. If we can find such numbers, then we continue the iteration with the new values.
  If there is only one number x left between a and b, we calculate the corresponding query cost and compare it with $cost_b$. The number that leads to the minimal cost will be the number of simple predicates that we need for horizontal fragmentation, i.e. for $cost_x < cost_b$ we choose $y := x$, otherwise $y := b$.
  If there is no number between a and b left in X, then b is the number of simple predicates needed for fragmenting C_i. Thus, $y := b$.

The result is the set $\Phi_i^y = \{\varphi_{i1}, \ldots \varphi_{iy}\}$ of simple predicates. Take the corresponding set $\mathcal{N}^y$ of normal predicates for fragmenting C_i.
Note that a, b, x_1, x_2 are updated in each loop. Searching requires only one or two new points in each loop. Only for these new points, query costs have to be calculated. The procedure stops when the remaining interval (a, b) only contains one number or is empty.

The above procedure is a heuristic one based on the assumption that a reasonable fragmentation schema can be obtained by looking at most frequently used simple predicates. Our approach with the above steps can rapidly search for a reasonable number of simple predicates which results in presumably low total query costs.

3.5 Analysis

In this section horizontal fragmentation for object oriented databases has been discussed in more detail. Definitions for simple predicates and normal predicates have been outlined. A heuristic horizontal fragmentation procedure has been proposed based on a cost model. The horizontal fragmentation approach presented in this section paper is superior to that presented in [13] and the one by [6]. Characteristics and potential benefits of the approach presented in this section, and differences to other approaches in the literature can be summarized as follows:

Firstly, the approach can deal with both simple and complex attributes using the same procedure. In [6], simple and complex attributes are treated with different algorithms. Simple attributes refer to the attributes of primitive attribute types only, i.e. those that do not contain other classes as part of them. Complex attributes have the domain of attribute as another class [6]. In our approach, attributes are defined using the underlying object oriented type system. Attributes defined on the abstract identifier type ID are treated as being defined on one of the base types. Our approach is more universal than others in terms of dealing with attributes of different types, and is more easily put into practical use.

Secondly, the expression of simple predicates is extended such that simple predicates can be defined on identifiers as well as on values. The expression of simple predicates on values is also extended to suit various type constructors in the underlying type system of the object oriented model. In [6], the definition of simple predicates is adopted from [13] without any adaption. However, the format of simple predicates in [13] cannot deal with the situation that a simple predicate is defined on a complex type constructor, i.e. a finite set type constructor. In the approach presented, the format of simple predicates is extended in a way that simple predicates can be defined not only on record type constructors or base type constructors but also on finite set type constructors. Note that the domain of the values for the comparison operator θ has been extended to include some set comparing operations. Path expressions have been introduced to refer to any elements of a class.

Thirdly, instead of using minterm predicates as found in [13] and [6], we introduced normal predicates on classes as the satisfiable minterm predicates. Horizontal fragmentation operations based on a set of normal predicates could therefore be guaranteed to satisfy the characteristics of fragmentation discussed in [13]. The approach did not rely on dependencies between simple queries as in [13], because these can hardly be determined. It is very hard to use these dependencies to determine the satisfiability of a conjunction of simple predicates, or to simplify them if they are satisfiable.

Fourthly, the approach applies a cost model for horizontal fragmentation design. In [13], horizontal fragmentation is performed without evaluating the overall system performance. We argue that the larger degree of fragmentation does not necessary lead to the better system overall performance. There ex-

ists a cut off point for the degree of horizontal fragmentation that the system has the best performance. However it is computationally intractable to find the optimized fragmentation solution by comparing total costs for all possible fragmentation schemata [13]. The heuristic procedure proposed in this section paper is based on a cost model with which the system performance can be evaluated once a database is being fragmented. One of the characteristics of this procedure is that it can rapidly achieve a horizontal fragmentation schema that is designed to result in low total query cost, or, in other words, the system's overall performance being improved.

4 Conclusion

In this paper we presented a heuristic approach to horizontal fragmentation for object oriented databases. The major objective is to provide a tractable approach to minimising the query processing costs for the most frequent queries. In general, this would require to consider all possible fragmentations and all possible allocations of intermediate query results to the nodes of a network, which is intractable. Instead of this we suggest to consider reducing the number of fragments by determining the right cut point in a list of simple predicates.

Our approach is based on a rather sophisticated data model, which in general needs more than horizontal and vertical fragmentation for distribution design. For this we introduced a third fragmentation operation called splitting. We are currently working on vertical and splitting fragmentation. Experimental evaluation for the proposed procedure is in its early phases. The work presented in this paper is our first attempt to overcome deficiencies of distribution design for object oriented databases. We consider the use of a simplistic data model and the neglection of intermediate query results as the most striking examples of such deficiencies.

References

1. Abiteboul, S., and Kanellakis, P. Object identity as a query language primitive. In *Proc. SIGMOD 1989* (1989), ACM.
2. Apers, P. M. G. Data allocation in distributed database systems. *ACM Trans. Database Syst. 13* (1988), 263–304.
3. Bellatreche, L., Karlapalem, K., and Simonet, A. Algorithms and support for horizontal class partitioning in object-oriented databases. *Distributed and Parallel Databases 8*, 2 (2000), 155–179.
4. Ceri, S., and Pelagatti, G. *Distributed Databases Principles and System.* McGraw-Hill, New York, 1984.
5. Chinchwadkar, G. S., and Goh, A. An overview of vertical partitioning in object oriented databases. *The Computer Journal 42*, 1 (1999).
6. Ezeife, C. I., and Barker, K. A comprehensive approach to horizontal class fragmentation in a distributed object based system. *Distributed and Parallel Databases 3*, 3 (1995), 247–272.

7. EZEIFE, C. I., AND BARKER, K. Distributed object based design: Vertical fragmentation of classes. *Distributed and Parallel Databases 6*, 4 (1998), 317–350.
8. KHALIL, N., EID, D., AND KHAIR, M. Availability and reliability issues in distributed databases using optimal horizontal fragmentation. In *Database and Expert Systems Applications* (1999), T. J. M. Bench-Capon, G. Soda, and A. M. Tjoa, Eds., vol. 1677 of *Lecture Notes in Computer Science*, Springer, pp. 771–780.
9. MA, H. Distribution design in object oriented databases. Master's thesis, Massey University, 2003.
10. MA, H., AND SCHEWE, K.-D. Fragmentation of XML documents. In *Proceedings XVIII Simpósio Brasileiro de Bancos de Dados (SBBD 2003)* (Manaus, Brazil, 2003), pp. 200–214.
11. MALINOWSKI, E., AND CHAKRAVARTHY, S. Fragmentation techniques for distributing object-oriented databases. In *Conceptual Modeling - ER '97* (1997), D. W. Embley and R. C. Goldstein, Eds., vol. 1331 of *Lecture Notes in Computer Science*, Springer, pp. 347–360.
12. NAVATHE, S. B., AND RA, M. Vertical partitioning for database design: A graphical algorithm. *ACM SIGMOD 14*, 4 (1989), 440–450.
13. ÖZSU, M. T., AND VALDURIEZ, P. *Principles of Distributed Database Systems*. Alan Apt, New Jersey, 1999.
14. RA, M. Horizontal partitioning for distributed database design. In *Advances in Database Research* (1993), M. Orlowska and M. Papazoglou, Eds., World Scientific Publishing, pp. 101–120.
15. SCHEWE, K.-D. On the unification of query algebras and their extension to rational tree structures. In *Proc. Australasian Database Conference* (2001), M. Orlowska and J. Roddick, Eds.
16. SCHEWE, K.-D. Fragmentation of object oriented and semi-structured data. In *Databases and Information Systems II* (2002), H.-M. Haav and A. Kalja, Eds., Kluwer Academic Publishers, pp. 1–14.
17. SCHEWE, K.-D., AND THALHEIM, B. Fundamental concepts of object oriented databases. *Acta Cybernetica 11*, 4 (1993), 49–84.
18. TAMHANKAR, A. M., AND RAM, S. Database fragmentation and allocation: An integrated methodology and case study. *IEEE Transactions on Systems Management 28*, 3 (1998), 194–207.
19. ZHANG, Y. On horizontal fragmentation of distributed database design. In *Advances in Database Research* (1993), M. Orlowska and M. Papazoglou, Eds., World Scientific Publishing, pp. 121–130.

SCIENTIFIC PAPERS UNIVERSITY OF LATVIA. 2004. VOLUME 672, Datbases and Information Systems
47–61 P.

Successful Database Integration through View Cooperation

Vojtech Vestenický

Charles University, Prague
Malostranské nám. 25, 118 00 Praha 1, Czech Republic
vestenicky@ksi.ms.mff.cuni.cz

Abstract. We present an approach for cooperation between databases modelled as HERM schemas. Cooperation between semantically related parts is based on views. Such views transparently support an information exchange between schemas. View definitions are generated in form of HERM algebra expressions by rules. The rules aim at solving integration conflicts, including constraints, on schema and instance level. The advantage is their simplicity and low complexity (i.e. polynomial). They can be combined into new rules to cover more complicated cases. Resulting views in HERM algebra can be translated into SQL and deployed on any common DBMS.

1 Introduction

Full view integration aims at developing a more general schema and a set of selectors on this schema such that each view is generated by a selector. The drawback is the undecidability of view integration in general and the infeasibility of view integration in most cases [Con86]. The advantage is that it provides a general solution if there is a solution.

View consistency tries to find the mechanisms which can be used for the cooperation or co-existence of views. View cooperation enables exchange of information between views. The disadvantage is the need for computational support. The advantage is the existence of view cooperation in situations where view integration is unlikely [Tha01].

Our approach proposes a cooperation strategy. We argue that it has several advantages over the approaches aiming at full view integration:

In global as view approaches (GAV), like COIN [GBMS99], MOMIS [BCVB01] and IBIS [CCG+03], the complexity is high, constraints either simple or omitted and there is no support for complex types.

In local as view approaches (LAV) [Lev00,Hal01,Len02] we face the same problems as in GAV.

The combination of the GAV and LAV introduced in [PA03] as Both As View (BAV) approach uses reversible schema transformations. The approach concentrates only on schema related conflicts and neglects the constraints.

A well known approach to solving structural conflicts between schemas using view integration has been proposed in [SP94]. Transformation rules produce new views that can be used to obtain an integrated schema. Our approach resembles above Spaccapietra's work in some points: we use rules to generate views, we constructs the functions to map values and consider constraints. However, the complexity of generating the views in our approach is lower (especially in rules including paths). We do not modify the views to construct an integrated schema, because it is not always possible and often not desired. We distinguish two types of views: query views and updatable views. This separation allows for maximal information extraction and use

of different update strategies. The cooperation relates to interoperable database systems. It is defined by functions which map one database system into another. Two generated views may support different operations, e.g. for an update there can exist mapping only in one direction. This offers much more flexibility. The success of the integration process is closely related to a data model and modeling techniques [VLF00]. Poor modeling techniques result in poor models which are very hard to integrate. We have chosen a Higher-order Entity-Relationship Model (HERM) [Tha00] to represent all participating schemas. HERM has advantages over the traditional ER Model and carries more semantic. Thus, we are able to handle complex types and more constraints than other approaches.

We now briefly introduce some of HERM extensions:

For pairwise different nested attributes $X_1, \ldots, X_n$ is $X(X_1, \ldots, X_n)$ a *nested tuple attribute*.

If Y is a nested attribute then $X\{Y\}$ is a *nested set attribute*.

We denote a set of attributes of some type R (R can be also the whole schema) with $attr(R)$, a set of entity types $ent(R)$ and a set of relationship types $rel_i(R)$, where i is the order of the relationship types (e.g. $ent(R) = rel_0(R)$). A set of components of a relationship type R is $comp(R)$.

A *cluster type* $C = R_1 + R_2 + \ldots + R_k$, $C^C = \bigcup_{i=1}^{k} R_i^C$ (C^C and R^C are instances of a cluster and relationship types respectively), where $\forall R_i^C, R_j^C,\ i \neq j, i, j = 1, \ldots, k$, $R_i^C \cap R_j^C = \emptyset$. If $R_1, \ldots, R_k$ are entity types then C is a cluster of entity types. The cluster type models the super/subtype relationship with more than one subtype. We require that identification types of the components of the cluster are domain-compatible.

The semantic correspondence between R_1 and R_2, entity/relationship types from different schemas, is given by a relation $\mathcal{E}(R_1, R_2)$.

For the rest of the paper we assume two different schemas S_1 and S_2 as an input to the integration process. The relation $\mathcal{E}$ is defined only for the elements of these schemas. We further require that both schemas S_1 and S_2 are connected (see Def. 1).

Definition 1 *Schema $S = \{E_1, \ldots, E_m, R_1, \ldots, R_n\}$ is called connected, if all relationship types have only entity types $E_1, \ldots, E_m$ and relationship types $R_1, \ldots, R_n$ as their components.*

For the definition of views we use *HERM algebra operations*: σ_ϕ (selection) with a selection formula ϕ, $\pi_{A_1,\ldots,A_m}$ (projection) with a generalized subset $\{A_1, \ldots, A_m\}$, ρ_f (renaming) with a renaming function f, $\bowtie_G$(join) with a common generalized subset G, $\cup$ (union), – (difference), $\nu_{X:\{A_1,\ldots,A_n\}}$(nest) and μ_A (unnest) with a set attribute A (see [Tha00] for details).

HERM diagram is a graphical representation of the HERM model. The clusters are represented with a symbol $\oplus$. The edges $E(ntity) \rightarrow A(ttribute)$ can be labelled by $dom(A)$. Other edges could be labelled by integrity constraints, e.g. cardinality constraints. All attributes A of a $R(elationship)$ constituting a key, i.e. $A \in id(R)$, are underlined.

This approach builds on the theory introduced in [Tha00], where the integration process is described as follows: Given the two database schemas S_1, S_2 and the corresponding database states S_1^C, S_2^C.

1. We must find the functions f_{12} and f_{21} which realize the partial embedding of types from schema S_1 to S_2 and S_2 to S_1, i.e. $f_{12} : S_1 \rightarrow\!\!\!\!\circ\ S_2$, $f_{21} : S_2 \rightarrow\!\!\!\!\circ\ S_1$.
2. Then the functions f_{12}^C and f_{21}^C need to be constructed which define a partial embedding of corresponding instances from S_1^C to S_2^C, i.e. $f_{12}^C : S_1^C \rightarrow\!\!\!\!\circ\ S_2^C$, $f_{21}^C : S_2^C \rightarrow\!\!\!\!\circ\ S_1^C$.

The two schema morphisms f_{12}, f_{12}^C and f_{21}, f_{21}^C cooperate and define a view cooperation (see Fig. 1) if for each $T_1 \in S_1^V \cap f_{21}(S_2^V)$ and each $T_2 \in S_2^V \cap f_{12}(S_1^V)$, for each pair $T_1^C \in S_1^C$ and $T_2^C \in S_2^C$

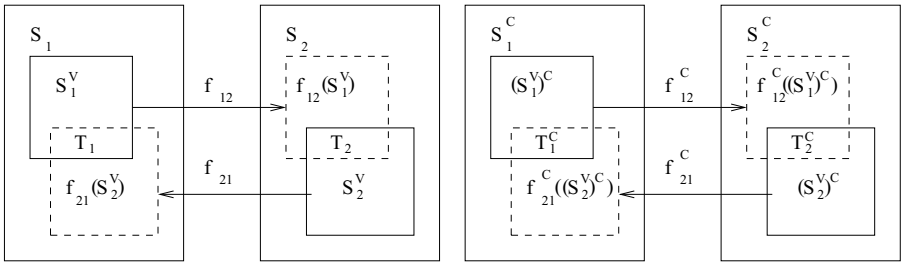


Fig. 1. View Cooperation

- the functions $f_{12}^C(T_1^C)$,$f_{21}^C(f_{12}^C(T_1^C))$,$f_{21}^C(T_2^C)$,$f_{12}^C(f_{21}^C(T_2^C))$ are defined and
- the equalities $f_{21}^C(f_{12}^C(T_1^C)) = T_1^C$, $f_{12}^C(f_{21}^C(T_2^C)) = T_2^C$ are valid.

The functions f_{12} and f_{21} are in our approach represented by the view definitions in HERM algebra.

2 Integration process

It is comprised of the following main phases:

1. **Obtain views for queries**: application of rules and generation of view definitions using the HERM algebra for each pair of semantically corresponding components in $\mathcal{E}$, no constraints are considered. The result is a maximal set of views containing semantically related components and their attributes.
2. **Obtain views for updates**: rules generate updatable views by adjusting the query views to satisfy constraints.
3. **Obtain mapping functions for instances**: construction of mapping functions for the view instances.
4. **Obtain cooperating views**: searching for overlapping parts in the pairs of the views. These parts represent then the final result of the integration process, cooperating views.
5. **Translation into SQL**: the resulting view definitions are translated into SQL statements that create the actual views in the underlying database.

There are some typical cases which cover common conflicts. They are related not only to the construction of the rules, but also to the construction of the views. View definitions in our approach can be limited to two *view patterns*:

- The views representing the semantic correspondence of two entity/relationship types and all their semantically related attributes are contained within these types, look like $\pi_{attr}(\rho_f(R))$, where $attr$ is a list of semantically related attributes, function f solves the naming inequalities by defining the mapping from one name space to another and R is the relation from which the mapping starts.
- Structural incompatibilities are the main reason for the existence of these view types. If the semantically related components are in different relationship types then it is necessary to access those types and extract the desired information. The view definition must therefore contain a join: $\pi_{attr}(\rho_f(R_1 \bowtie (\ldots \bowtie (R_{n-1} \bowtie R_n))))$, where $\{R_1, \ldots, R_n\}$ is a set of relationship types that are connected and have the semantically related attributes as their components.

2.1 Obtain views for queries

These views contain the maximal possible set of semantically related components. They aim at information extraction and may not be updatable. They serve as building components for further view definitions. The views satisfying the restrictions posed by different schemas are a subset of these query views, i.e. theirs specialization.

Views are generated by rules. The rules expect as an input two types from different schemas S_1 and S_2. They can be entity/relationship types and must satisfy the condition stated in the beginning of the rule. These conditions allow to handle different types of integration problems. We mention only a few rules for illustration (for others see [Ves03]).

1. **For entity types one of them with nested set attribute** $E_1 \in S_1$ and $E_2, E_3, R \in S_2$ where:
   - $E_1 = (A_1, \ldots, A_n, A\{X\})$, $A_1, \ldots, A_n$ are simple attributes, $A\{X\}$ is a nested set attribute containing the type X,
   - $id(E_1) \in attr(E_1) - \{A\}$, i.e. nested attribute is not a part of the key,
   - $E_2 = (B_1, \ldots, B_k)$, $(E_1, E_2) \in \mathcal{E}$, i.e. E_1 and E_2 are semantically related,
   - $E_3 = (C_1, \ldots, C_l)$, $\exists c \in attr(E_3).(c, A) \in \mathcal{E}$, i.e. for the complex attribute A of E_1 there exists some semantically corresponding attribute c in E_3,
   - $comp(R) = \{E_2, E_3\}$, i.e. E_2 and E_3 are components of the relationship type R.

   This rule handles the case where an entity type in the first schema contains a nested set attribute and in the second schema there exists a decomposition into two entity types one of them holding the equivalent of the nested attribute.

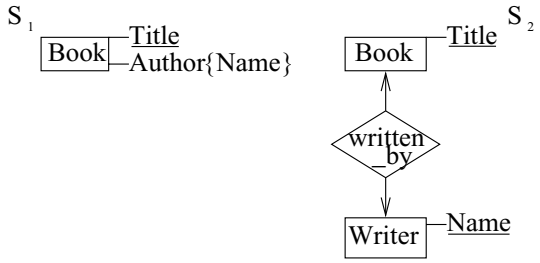


Fig. 2. Entity $Book$ with the nested set attribute $Author$ and the decomposed equivalent in schema S_2

We explain the situation using the Fig. 2. The entity type $Book$ in the schema S_1 has a nested set attribute $Author$ containing the set of elements of type $Name$. This simple example reflects the fact, that a book may have been written by more authors and we want to keep the track of their names. In schema S_2 can the same situation be modelled using the entity type $Book$ and a separate entity type $Writer$ with the attribute $Name$. Both entity types of schema S_2 are connected through the relationship type $written\_by$. We define the views for this example first. Let us assume $\mathcal{E}$ contains the semantic correspondences between components: $\mathcal{E} = \{(Book, Book), (Author, Name), (Title, Title)\}$. The views are then:

– $V_{S_1.Book, S_2.Book} = \pi_{Title, Name}(\rho_{Author \to Name}(S_1.Book))$,

– $V_{S_2.Book,S_1.Book} = \pi_{Title,Author}(\rho_{Name \to Author}(S_2.Book \bowtie (written\_by \bowtie Writer)))$.

Note: the attribute $Name$ in the projection part of the view $V_{S_1.Book,S_2.Book}$ is a complex attribute and holds the set of author names. We must therefore transform this complex attribute before we can work with instances, see Sec. 2.3.

Having defined the views in the above example, we turn to the general case and construct:

– *the mapping functions for the attributes*:
$map_{E_1,E_2} = \{(a,b)|a \in attr(E_1), b \in attr(E_2) \cup attr(E_3), (a,b) \in \mathcal{E}\}$,
$map_{E_2,E_1} = \{(a,b)|a \in attr(E_2) \cup attr(E_3), b \in attr(E_1), (a,b) \in \mathcal{E}\}$.

– *view definitions* using the HERM algebra:
- $aset_{E_1,E_2} = \{b|(a,b) \in map_{E_1,E_2}\}$,
$V_{E_1,E_2} = \pi_{aset_{E_1,E_2}}(\rho_{map_{E_1,E_2}}(E_1))$
- $aset_{E_2,E_1} = \{b|(a,b) \in map_{E_2,E_1}\}$,
$V_{E_2,E_1} = \pi_{aset_{R_2,R_1}}(\rho_{E_2,E_1}(E_2 \bowtie (R \bowtie E_3)))$

Other structural conflicts involving complex types can be solved by similar rules.

2. **For two relationship types $R_1 \in S_1, R_2 \in S_2$ and semantically related attributes are not contained within the matching relationship type** where:
- $R_1 = (comp(R_1), A_1, \ldots, A_n)$,$R_2 = (comp(R_2), B_1, \ldots, B_m)$ and $(R_1, R_2) \in \mathcal{E}$.

Here the attributes can have their semantic equivalents in different relationship/entity types. See Fig. 3, e.g. the semantically corresponding attributes A' are outside the matching type R_1. The dotted lines represent the semantic correspondence between the attributes and relationship types. We determine the set of corresponding attributes for R_1 and R_2:
$aset^{e\_attr}_{R_1} = \{B_j|A_i \in attr(R_1), i = 1, \ldots, n, (A_i, B_j) \in \mathcal{E}\}$
$aset^{e\_attr}_{R_2} = \{A_i|B_j \in attr(R_2), j = 1, \ldots, m, (A_i, B_j) \in \mathcal{E}\}$

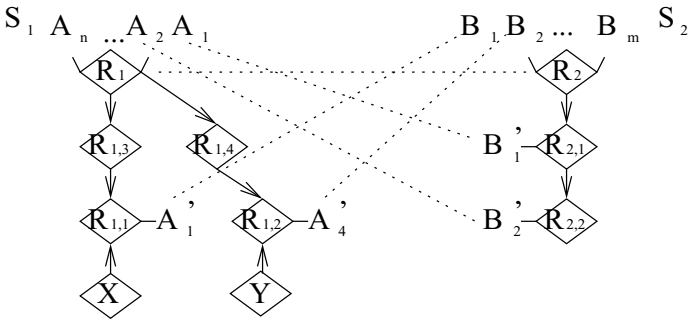


Fig. 3. Related attributes are outside the matching types R_1 and R_2

Then we find relationship types where these attributes reside, e.g. in Fig. 3 $R_{1,1}$,$R_{1,2}$.

Definition 2 *Function* $f_{rel} : A \to P$, $A \in attr(S)$, $P \in rel(S)$ *is defined as* $f_{rel}(A) = \{p|a \in A, a \in attr(p)\}$.

Assume we get the sets of relationship types for both attribute sets: $rset_{R_1}^{rel} = f_{rel}(aset_{R_2}^{e\_attr})$ and $rset_{R_2}^{rel} = f_{rel}(aset_{R_1}^{e\_attr})$. For each of the above sets other types must be found, such that all types are connected through some path. Those nodes must exist in each schema because the schemas are connected. If we look at the Fig. 3 we can see the types, e.g. $R'_{1,3}$ and $R'_{1,4}$. We define the function that returns those extended sets using partially an algorithmic description:

Definition 3 *The function $find : R \times R' \rightarrow R''$, where R is an input set of relationship types, R' is a set of all relationship types of schema S, and R'' is a result set of connected relationship types containing R.*

function *find*$(rset, rschema)$
$Con = \{r | r \in rset, s \in rset - \{r\}, r \in comp(s) \vee s \in comp(r)\}$ `/** already connected`
$Con\_r = \{r | r \in Con, \nexists s \in Con - \{r\}.r \in comp(s)\}$ `/** root types`
$Con\_s = Con - Con\_r$
$rset\_r = rset - Con\_r$
if $(Con\_r = rset\_r)$ **then return** $rset$ `/** check if all connected`
else
begin
$new\_r = \{s | r \in rset\_r, s \in rschema, r \in comp(s)\}$ `/** extend the set`
$new\_input = new\_r \cup rset\_r$
find=$Con\_s \cup$ *find*$(new\_input, rschema)$ `/** check new extended set`
end

Theorem 1 *Function find terminates and its complexity is polynomial, i.e. $O(3n^3)$, where $n = |rschema|$ (number of schema relationship types).*

The proof is straightforward. The function $find$ may return some types which are not needed to keep the required components connected (see Fig. 3 X and Y). The following function removes unnecessary types from the resulting set.

Definition 4 *Function $remove : I \times R \rightarrow R'$, where I equals the parameter rset of function find, R is the result from find, and R' is a resulting set of relationship types without unnecessary types.*

function *remove*$(Input, Result)$
$Test = Result - Input$
while $|Test| > 0$ **do**
$t \in Test$
$IsCon = Result - \{t\}$ `/** try to remove one`
if $(\forall x, y \in IsCon.x \neq y \Rightarrow (x \in comp(y) \vee y \in comp(x)))$ **then** `/** still connected?`
$Result = Result - \{t\}$ `/** OK, remove it`
$Test = Test - \{t\}$
do while
return $Result$

Theorem 2 *Function remove terminates and its complexity is $O(n^3)$, where $n = |Test|$.*

Assume we executed the $find$ and $remove$ functions:

$Result_{R_1} = find(rset^{rel}_{R_1}, rel(S_1)), rset^{Con}_{R_1} = remove(rset^{rel}_{R_1}, Result_{R_1})$,
$Result_{R_2} = find(rset^{rel}_{R_2}, rel(S_2)), rset^{Con}_{R_2} = remove(rset^{rel}_{R_2}, Result_{R_2})$
and received the results of the form: $rset^{Con}_{R_1} = \{R_1, R_{11}, \ldots, R_{1c}\}$, $rset^{Con}_{R_2} = \{R_2, R_{21}, \ldots, R_{2d}\}$. Notice that each of the previous sets have the matching relationship type R_1 or R_2 as their element. We further need *root types* of these sets.

Definition 5 *Function* $root\_types : R \rightarrow R'$*, where* R *is a set of relationship types and* R' *is a set of root types,* $root\_types = \{r | r \in R, \nexists s \in R - \{r\}.r \in comp(s)\}$.

Theorem 3 *Function* $root\_types$ *runs in* $O(n^2)$*, where* $n = |rset|$.

Suppose we executed the function: $rset^{Root}_{R_1} = root\_types(rset^{Con}_{R_1})$, $rset^{Root}_{R_2} = root\_types(rset^{Con}_{R_2})$ and obtained the sets: $rset^{Root}_{R_1} = \{R_1, R_{11}, \ldots, R_{1e}\}, rset^{Root}_{R_2} = \{R_{21}, \ldots, R_{2f}\}$.
We now have the root types in $rset^{Root}_{R_i}$ and the connected types in $rset^{Con}_{R_i}$, for $i = 1, 2$. It is further valid that $rset^{Root}_{R_i} \subseteq rset^{Con}_{R_i}$. We separate the non-root types out of the connected set of types: $rset^{\overline{Root}}_{R_i} = rset^{Con}_{R_i} - rset^{Root}_{R_i}$. Let those respected sets be: $rset^{\overline{Root}}_{R_1} = \{R_{1,p}, \ldots R_{1,r}\}, rset^{\overline{Root}}_{R_2} = \{R_{2,s}, \ldots R_{2,t}\}$. The root types will be used in the join part of the view first, then come the non-root types. This is based on the assumption that the root types possess also the keys of the non-root types.

Theorem 4 *For the root types in a set of* $rset^{Root}_{R_i}$ *there exists a sequence* $Seq = \langle r_1, \ldots, r_n \rangle$*, where* $n = |rset^{Root}_{R_i}|$ *such that:*
$\forall r_j \in rset^{Root}_{R_i}, j = 1, \ldots, n, i = 1, 2.((id(r_1) \subseteq id(r_2)) \vee (id(r_2) \subseteq id(r_1))) \wedge ((id(r_2) \subseteq id(r_3)) \vee (id(r_3) \subseteq id(r_2))) \wedge \ldots \wedge ((id(r_{j-1}) \subseteq id(r_j)) \vee (id(r_j) \subseteq id(r_{j-1}))) \Rightarrow \langle r_1, \ldots, r_n \rangle$.

Proof: the existence of a sequence Seq is guaranteed by the connectedness of the input schemas.
We need this sequence for the join part of the view where the relationship types being joined should have some common set of joining attributes.

Definition 6 *Function* $root\_seq : R \rightarrow (R', I)$ *gets as an input parameter the set of root types* R *and returns the set of pairs* (R', I) *where* R *is a root type and* I *is the order in the sequence.*

function *root_seq(Root)*
$r \in Root$
$Key = id(r)$ `/** current set of keys`
$i = 1$
$Result = Result \cup \{(r, i)\}$ `/** first in sequence`
$Root = Root - \{r\}$
while $(|Root| > 0)$ **do**
 for $s \in Root$ **do** `/** find next`
 if $((Key \cap id(s)) \neq \emptyset)$ **then** `/** check if common keys exist`
 begin
 $i = i + 1$
 $Result = Result \cup \{(s, i)\}$ `/** next in the sequence found`
 $Key = Key \cup id(s)$ `/** extend the key set`
 $Root = Root - \{s\}$
 end

do for
do while
return $Result$

Theorem 5 *Complexity of the function* $root\_seq$ *is* $O(n^2)$*, where* $n = |Root|$.

We now have all what we need to define the views:

- *the mapping functions for the attributes*:
  $map_{R_1,R_2} = \{(a,b)|a \in aset_{R_2}^{e\_attr}, b \in R_2, (a,b) \in \mathcal{E}\}$,
  $map_{R_2,R_1} = \{(a,b)|a \in aset_{R_1}^{e\_attr}, b \in R_1, (a,b) \in \mathcal{E}\}$,
  the attribute sets for the projection part:
  $aset_{R_1} = \{b|(a,b) \in map_{R_1,R_2}\}$,$aset_{R_2} = \{b|(a,b) \in map_{R_2,R_1}\}$,
- *view definitions*:
  Suppose we executed the function $root\_seq$ and got the following sets:
  $Seq_{R_1}^{Root}$ =root_seq($rset_{R_1}^{Con}$),$Seq_{R_1}^{Root} = \{(R_{1,1},1),\ldots,(R_{1,g},g)\}$,
  $Seq_{R_2}^{Root}$ =root_seq($rset_{R_2}^{Con}$),$Seq_{R_2}^{Root} = \{(R_{2,1},1),\ldots,(R_{2,h},h)\}$.
  The types with the lower index are used in the join definition first. Then we have non-root types: $rset_{R_1}^{\overline{Root}} = \{R_{1,p},\ldots R_{1,r}\}$, $rset_{R_2}^{\overline{Root}} = \{R_{2,s},\ldots R_{2,t}\}$.
  - $V_{R_1,R_2} = \pi_{aset_{R_1}}(\rho_{map_{R_1,R_2}}(R_{1,p} \bowtie (\ldots \bowtie (R_{1,r} \bowtie (R_{1,g} \bowtie (\ldots \bowtie (R_{1,2} \bowtie R_{1,1})))))))$
  - $V_{R_2,R_1} = \pi_{aset_{R_2}}(\rho_{map_{R_2,R_1}}(R_{2,s} \bowtie (\ldots \bowtie (R_{2,t} \bowtie (R_{2,g} \bowtie (\ldots \bowtie (R_{2,2} \bowtie R_{2,1})))))))$

Rules solving other conflicts, e.g. in IsA hierarchies, can be easily derived from this rule.

3. **Entity/relationship type with entity/relationship type containing cluster** $R_1 \in S_1$ and $R_2 \in S_2$ where:
   - $R_1 = (comp(R_1), A_1, \ldots, A_n)$, if $comp(R_1) = \emptyset$ then R_1 is an entity type,
   - $R_2 = (comp(R_2), R_{2,1} + \ldots + R_{2,s}, B_1, \ldots, B_m)$, $R_2 \in rel_i(S_2)$, R_2 contains cluster
     $R_{2,1} + \ldots + R_{2,s}$, where $\{R_{2,1} + \ldots + R_{2,s}\} \in rel_j(S_2), j < i$,
   - $(R_1, R_2) \in \mathcal{E}$, i.e. R_1 is semantically related to R_2

These conflicts can arise, if one designer uses just one type or different degree of abstraction, and another prefers super/subtype relationships using clusters. The result of such process may look like in Fig. 4. Here the information about employees has been differently modelled. In schema S_1 only one entity type has been chosen: $Employee(SSN, Name, Family\_name, Begin, End, Position, Project, id\{SSN\})$. Schema S_2 contains the relationship type $Employee$ with the cluster: $Employee(Lecturer{+}Project\_assistant, First\_name, Surname, C\_Begin, C\_End, id\{Lecturer{+}Project\_assistant\})$. The key SSN from the cluster has been inherited.

Let us assume that the semantic correspondences are defined by the relation $\mathcal{E} = \{(SSN,SSN),(Position,Type),(Project,Project),(Name,First\_name),(End,C\_End),$
$(Family\_name,Surname),(Begin,C\_Begin)\}$. The cluster is disjoint union of types. We will therefore define the views for every pair $Employee$ and its cluster component of schema S_2, i.e. $(Employee, Lecturer)$ and $(Employee, Project\_assistant)$. The view definitions are then:

- $V_{S_1.Employee,S_2.Employee-Lecturer} = \pi_{First\_name,Surname,C\_Begin,C\_End,Type}(\rho_{Name\to First\_name,Family\_name\to Surname,Begin\to C\_Begin,End\to C\_End,Pos.\to Type}(S_1.Employee))$,
- $V_{S_2.Employee-Lecturer,S_1.Employee} = \pi_{SSN,Name,Family\_name,Begin,End,Position}(\rho_{First\_name\to Name,Surname\to Family\_name,C\_Begin\to Begin,C\_End\to End,Type\to Pos.}(Lecturer \bowtie S_2.Employee))$,

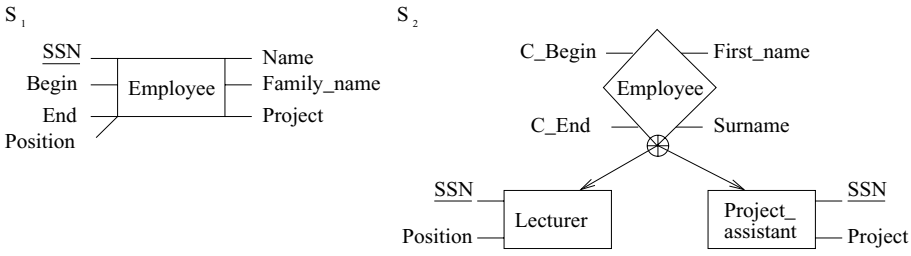


Fig. 4. The entity type *Employee* in S_1 is modelled as cluster in S_2

- $V_{S_1.Employee,S_2.Employee-Project\_assistant} =$
  $\pi_{First\_name,Surname,C\_Begin,C\_End,Project}$
  $(\rho_{Name \to First\_name,Family\_name \to Surname,Begin \to C\_Begin,End \to C\_End}$
  $(S_1.Employee))$,
- $V_{S_2.Employee-Project\_assistant,S_1.Employee} =$
  $\pi_{SSN,Name,Family\_name,Begin,End,Project}$
  $(\rho_{First\_name \to Name,Surname \to Family\_name,C\_Begin \to Begin,C\_End \to End}$
  $(Project\_assistant \bowtie S_2.Employee))$.

The view definitions for a general case must also reflect the different choices:

- *the mapping functions for the attributes*, where $t = 1, \ldots, s$:
  $map_{R_1,R_2-R_{2,t}} = \{(a,b) | a \in attr(R_1), b \in attr(R_2) \cup attr(R_{2,t}), (a,b) \in \mathcal{E}\}$,
  $map_{R_2-R_{2,t},R_1} = map_{12}^{-1}$,

- *view definitions*,where $t = 1, \ldots, s$:
  - $aset_{R_1,R_2-R_{2,t}} = \{b | (a,b) \in map_{R_1,R_2-R_{2,t}}\}$,
    $V_{R_1,R_2-R_{2,t}} = \pi_{aset_{R_1,R_2-R_{2,t}}}(\rho_{map_{R_1,R_2-R_{2,t}}}(R_1))$,
  - $aset_{R_2-R_{2,t},R_1} = \{b | (a,b) \in map_{R_2-R_{2,t},R_1}\}$,
    $V_{R_2-R_{2,t},R_1} = \pi_{aset_{R_2-R_{2,t},R_1}}(\rho_{map_{R_2-R_{2,t},R_1}}(R_{2,t} \bowtie R_2))$.

2.2 Obtain views for updates

In this phase of the integration process another rules check the views against the most important constraints, key constraints, attribute constraints and domain incompatibilities (other can be easily added) to make the views updatable. We select the rule from Sec. 2.1 and consider the updatability.

For two relationship types where semantically related attributes are not contained within the matching relationship type. The query view definitions are characterized by two types of sets of relationship types used in the join parts of the views: root types and non-root types. By the root types plays the ordering an important role: $Seq_{R_1}^{Root} = \langle (R_{1,1}, 1), \ldots, (R_{1,g}, g) \rangle$, $Seq_{R_2}^{Root} = \langle (R_{2,1}, 1), \ldots, (R_{2,h}, h) \rangle$, $x \in Seq_{R_i}^{Root}$ means that an element x is a part of the sequence $Seq_{R_i}^{Root}$. Let the non-root types be: $rset_{R_1}^{Root} = \{R_{1,p}, \ldots R_{1,r}\}$, $rset_{R_2}^{Root} = \{R_{2,s}, \ldots R_{2,t}\}$. The view definitions are:

- $V_{R_1,R_2} = \pi_{aset_{R_1}}(\rho_{map_{R_1,R_2}}(R_{1,p} \bowtie (\ldots \bowtie (R_{1,r} \bowtie (R_{1,g} \bowtie (\ldots \bowtie (R_{1,2} \bowtie R_{1,1})))))))$,

– $V_{R_2,R_1} = \pi_{aset_{R_2}}(\rho_{map_{R_2,R_1}}(R_{2,s} \bowtie (\ldots \bowtie (R_{2,t} \bowtie (R_{2,g} \bowtie (\ldots \bowtie (R_{2,2} \bowtie R_{2,1})))))))$.

We follow the identifiability property and turn to the key attributes first. In order to update the views, the domains of these keys must be compatible.

Definition 7 *The function* $f^{comp\_keys} R_1 \times R_2 \rightarrow A$, $f^{comp\_keys}(R_1, R_2) = \{b | a \in id(R_1), b \in id(R_2), (a,b) \in \mathcal{E}, dom(a) = dom(b)\}$ *returns for the relationship type* R_1 *the set of semantically equal and domain compatible key attributes of relationship type* R_2.

There are two update strategies for these views:

1 if not all of the keys in R_1 have their semantically related and domain compatible pairs in R_2, i.e. $id(R_1) \supset f^{comp\_keys}_{R_2,R_1}$ (see Def. 7), then we say that the view V_{R_2,R_1} is not updatable. The above condition is also valid for the proper parameters of view V_{R_1,R_2}. It means, it may happen that only one of the views is updatable while the other is not. Both views are updatable if the number of key attributes in both types R_1 and R_2 is equal and that is not the general case.

2 if all keys of R_1 have their semantically related and domain compatible pairs in R_2, i.e. $id(R_1) = f^{comp\_keys}_{R_2,R_1}$, then we say that the view V_{R_2,R_1} (respectively V_{R_1,R_2}) is updatable. But only for the attributes that reside in R_1 or their semantic counter parts in R_2. Attributes outside these types must be accessed through some relationship type. If we look at the Fig. 3, the components of the R_1 and R_2 (e.g. $R_{1,3}$,$R_{1,4}$) are identifiable, as their key sets are subsets of keys in R_1 and R_2. The problem is posed by other types, e.g. X and Y. If we want to update the attributes residing in these types, we must have the corresponding keys. We take the set of attributes, e.g. $aset_{R_1}$, from the original projection part of the view definition V_{R_1,R_2} (see above). This set contains the equivalent attribute pairs in schema S_2. We use the existing mapping map_{R_1,R_2} and extract the attributes $aset_{S_1}$ in schema S_1: $aset_{S_1} = \{a | (a,b) \in map_{R_1,R_2}\}$. So if we wanted to update the view V_{R_1,R_2} this is a set of attributes in schema S_1 which must be sufficient for the identification of all in the join definition participating relationship types. We do the same for the set $aset_{R_2}$ and get $aset_{S_2} = \{a | (a,b) \in map_{R_2,R_1}\}$. It remains to check for which relationship types these sets provide the identifiability. We consider root types first (keys of other types are included in root types).

Definition 8 *The function* $f^{ident}_{rel} : R \times A \rightarrow R'$ *for the set of relationship types* R *and attributes* A *returns the set of relationship types* R' *whose all key attributes are included in* A, *i.e.* $f^{ident}_{rel}(R, A) = \{r | r \in R, A_{id} \subseteq A, id(r) = A_{id}\}$.

We use the function f^{ident}_{rel}: $rset^{Root}_{R_1} = \{a | (a,b) \in Seq^{Root}_{R_1}\}$, $rset^{Root}_{R_2} = \{a | (a,b) \in Seq^{Root}_{R_2}\}$,
$rset^{Root\_ident}_{R_1} = f^{ident}_{rel}(rset^{Root}_{R_1}, aset_{S_1})$, $rset^{Root\_ident}_{R_2} = f^{ident}_{rel}(rset^{Root}_{R_2}, aset_{S_2})$.

While $rset^{Root\_ident}_{R_1} \subseteq rset^{Root}_{R_1}$ and $rset^{Root\_ident}_{R_2} \subseteq rset^{Root}_{R_2}$ holds, we must remove not only the components of unidentifiable root types but also those relationship types which are no longer connected with parts where R_1 respectively R_2 resides.

Definition 9 *The function* $f^{components}_{rel} : R \times R' \rightarrow R''$ *where* R *is a relationship type from which we want to determine the components (recursively to all sub-components),* R' *is a set of relationship types where the components of* R *may come from, and* R'' *is a resulting set containing the components of* R.

function $f_{rel}^{components}(R, R')$
for all $x \in R'$ **do**
if $x \in comp(R)$ **then return** $\{x\} \cup f_{rel}^{components}(x, R' - \{x\})$
do for

Definition 10 *The function* $f_{rel}^{new\_con} : R \times R_1 \times R_2 \rightarrow R_3$*, where* R *is a relationship type,* R_1 *is a set of root types,* R_2 *is a set of non-root types and* R_3 *is a new set of connected relationship types.*

function $f_{rel}^{new\_con}(R, root\_ident, non\_root)$
for all $r \in root\_ident$ **do**
$P = P \cup f_{rel}^{components}(r, non\_root)$
if $(f_{rel}^{components}(r, non\_root) \cap \{R\}) \neq \emptyset$ **then** $P_R = P_R \cup f_{rel}^{components}(r, non\_root)$
do for
$P_Z = P - P_R$
$Result = \emptyset$
while $|P_Z| > 0$ **do**
$q \in P_Z$
if $((\{q\} \cup Result) \cap P_R) \neq \emptyset$ **then** $Result = Result \cup \{q\}$
$P_Z = P_Z - \{q\}$
do while
return *Result*

We illustrate the usage of the above function. We have the sets of root types $rset_{R_1}^{Root\_ident}$ and $rset_{R_2}^{Root\_ident}$ which can be identified (we have all the keys) with the sets of attributes $aset_{S_1}$ and $aset_{S_2}$ respectively. We have sets of non-root relationship types $rset_{R_1}^{\overline{Root}}$ and $rset_{R_1}^{\overline{Root}}$. If we use these sets as input parameters to the function $f_{rel}^{new\_con}$ we get the following:
$rset_{R_1}^{new\_con} = f_{rel}^{new\_con}(R_1, rset_{R_1}^{Root\_ident}, rset_{R_1}^{\overline{Root}})$,
$rset_{R_2}^{new\_con} = f_{rel}^{new\_con}(R_2, rset_{R_2}^{Root\_ident}, rset_{R_2}^{\overline{Root}})$.
The results are two new connected sets for R_1 and R_2 under consideration of identifiability of root types using the semantically corresponding attributes. The ordering of root types for the join can be achieved by using the function $root\_seq$ for $rset_{R_1}^{Root\_ident}$ and $rset_{R_2}^{Root\_ident}$. Let those result sets be: $Seq_{R_1}^{Root\_ident} = \langle R_{1,1}, \ldots, R_{1,k} \rangle$, $Seq_{R_2}^{Root\_ident} = \langle R_{2,1}, \ldots, R_{2,l} \rangle$. We need new non-root types: $rset_{R_1}^{\overline{Root}} = rset_{R_1}^{new\_con} - rset_{R_1}^{Root\_ident}$, $rset_{R_2}^{\overline{Root}} = rset_{R_2}^{new\_con} - rset_{R_2}^{Root\_ident}$. Let those sets be: $rset_{R_1}^{\overline{Root}} = \{R_{1,p}, \ldots, R_{1,q}\}$, $rset_{R_2}^{\overline{Root}} = \{R_{2,r}, \ldots, R_{2,s}\}$.

The new sets of attributes for the projection part and mappings:
$aset_{R_1,R_2}^{new} = \{b | a \in attr(r), r \in rset_{R_1}^{new\_con}, (a,b) \in \mathcal{E}\}$,
$aset_{R_2,R_1}^{new} = \{b | a \in attr(r), r \in rset_{R_2}^{new\_con}, (a,b) \in \mathcal{E}\}$,

$map_{R_1,R_2}^{new} = \{(a,b) | a \in attr(r), r \in rset_{R_1}^{new\_con}, (a,b) \in \mathcal{E}\}$,
$map_{R_2,R_1}^{new} = \{(a,b) | a \in attr(r), r \in rset_{R_2}^{new\_con}, (a,b) \in \mathcal{E}\}$,
The new view definitions are of the form:

- $V_{R_1,R_2} = \pi_{aset_{R_1,R_2}^{new}}(\rho_{map_{R_1,R_2}^{new}}(R_{1,p} \bowtie (\ldots \bowtie (R_{1,q} \bowtie (R_{1,k} \bowtie (\ldots \bowtie (R_{1,2} \bowtie R_{1,1})))))))$
- $V_{R_2,R_1} = \pi_{aset_{R_2,R_1}^{new}}(\rho_{map_{R_2,R_1}^{new}}(R_{2,r} \bowtie (\ldots \bowtie (R_{2,s} \bowtie (R_{2,l} \bowtie (\ldots \bowtie (R_{2,2} \bowtie R_{2,1})))))))$

2.3 Obtain mapping functions for instances

In the previous sections we have constructed rules which actually map parts of one schema onto the parts of another schema using the view definitions. In the Fig. 1 it is the left part with the functions f_{12} and f_{21}. By, for example, domain incompatibilities the instances must be mapped between the schemas, we need the functions like f_{12}^C and f_{21}^C from the right-part of Fig. 1. We show some rules for illustration:

– **For two entity/relationship types** $R_1 \in S_1$ and $R_2 \in S_2$ where semantically equal attributes are contained within the matching entity/relationship type we have simple view definitions: $V_{R_1,R_2} = \pi_{aset_{R_1,R_2}}(\rho_{map_{R_1,R_2}}(R_1))$, $V_{R_2,R_1} = \pi_{aset_{R_2,R_1}}(\rho_{map_{R_2,R_1}}(R_2))$.
The update condition was the domain-compatibility of the key attributes of both types. This can be weakened, if there exist some function that maps the values from one domain into the values of another domain. We extend this principle to all attributes from the projection part of the view.
We find the sets of domain incompatible attributes for both views:
$rset_{R_1,R_2}^{dom\_incomp} = \{(a,b)|(a,b) \in map_{R_1,R_2}, dom(a) \neq dom(b)\}$,
$rset_{R_2,R_1}^{dom\_incomp} = \{(a,b)|(a,b) \in map_{R_2,R_1}, dom(a) \neq dom(b)\}$.
For these domain incompatible pairs we find the set of functions:
$fset_{R_1,R_2} = \{f|(a,b) \in rset_{R_1,R_2}^{dom\_incomp}, \exists f.f(dom(a)) = dom(b)\}$,
$fset_{R_2,R_1} = \{f|(a,b) \in rset_{R_2,R_1}^{dom\_incomp}, \exists f.f(dom(a)) = dom(b)\}$.
To guarantee the updatability of the views, we require that for all domain incompatible pairs of attributes from $rset_{R_1,R_2}^{dom\_incomp}$ and $rset_{R_2,R_1}^{dom\_incomp}$ such functions exist:
$\forall (a,b) \in rset_{R_1,R_2}^{dom\_incomp} \exists f \in fset_{R_1,R_2}.f(dom(a)) = dom(b)$,
$\forall (a,b) \in rset_{R_2,R_1}^{dom\_incomp} \exists f \in fset_{R_2,R_1}.f(dom(a)) = dom(b)$.
If some of the above conditions are not valid we must remove those attributes from the view definition for which there is no mapping function. The set of domain incompatible attributes of R_1 and R_2: $rset_{R_1}^{di} = \{a|(a,b) \in rset_{R_1,R_2}^{dom\_incomp}\}$, $rset_{R_2}^{di} = \{a|(a,b) \in rset_{R_2,R_1}^{dom\_incomp}\}$.
Let the sets of incompatible attributes for which there exist mapping functions be: $rset_{R_1}^{di} = \{A_j, \ldots, A_{j+k}\}$ and $rset_{R_2}^{di} = \{B_l, \ldots, B_{l+m}\}$. Let the functions for $rset_{R_1}^{di}$, $rset_{R_2}^{di}$ be: $fset_{R_1,R_2}^{dmap} = \{f_{11}, \ldots, f_{1k}\}$ and $fset_{R_2,R_1}^{dmap} = \{f_{21}, \ldots, f_{2m}\}$.
The mapping functions for the instances can then be defined like this:
$f_{R_1,R_2}^C : R_1^C[A_j] \times \ldots \times R_1^C[A_{j+k}] \rightarrow f_{11}(R_1^C[A_j]) \times \ldots \times f_{1k}(R_1^C[A_{j+k}])$,
$f_{R_2,R_1}^C : R_2^C[B_l] \times \ldots \times R_2^C[B_{l+m}] \rightarrow f_{21}(R_2^C[B_l]) \times \ldots \times f_{2m}(R_2^C[B_{l+m}])$.

– **For entity types one of them with nested set attribute** $E_1 \in S_1$ and $E_2, E_3, R \in S_2$ where the view definitions were (see Sec. 2.1):

- $aset_{E_1,E_2} = \{b|(a,b) \in map_{E_1,E_2}\}$, $V_{E_1,E_2} = \pi_{aset_{E_1,E_2}}(\rho_{map_{E_1,E_2}}(E_1))$,
- $aset_{E_2,E_1} = \{b|(a,b) \in map_{E_2,E_1}\}$, $V_{E_2,E_1} = \pi_{aset_{R_2,R_1}}(\rho_{E_2,E_1}(E_2 \bowtie (R \bowtie E_3)))$.

The entity type E_1 has a nested set attribute A. The entity type E_3 in other schema holds the semantically equal simple attribute C such that $(A,C) \in \mathcal{E}$. We try to solve the structuring problems with some suitable mapping between instances. We use the HERM algebra operations nest ν and unnest μ (for details refer to [Tha00]). The schema elements contained in the view V are denoted by $\mathcal{V}$. We can now introduce the mappings:
$f_{E_1,E_2}^C : \mu_C \pi_C(V_{E_1,E_2}^C) \rightarrow E_3^C[C]$,
$f_{E_2,E_1}^C : \nu_{A:\{C\}} \pi_C \rho_{A \rightarrow C}(V_{E_2,E_1}^C) \rightarrow E_1^C[A]$.

2.4 Obtain cooperating views

We have defined the conditions for the view cooperation in Sec. 1. We apply this criteria on the view definitions we have got by using the previous rules. We do not go into detail and suppose the views satisfy all constraints. We focus on the mappings. From the mappings we can decide, whether the current view definitions are cooperating or not. If not, we define the new views which are subsets of the old ones and satisfy the conditions for cooperation.

– **For two entity/relationship types** $R_1 \in S_1$, $R_2 \in S_2$ we had simple view definitions (see Sec. 2.3) and these mapping functions:
$map_{R_1,R_2} = \{(a,b)|a \in attr(R_1), b \in attr(R_2), (a,b) \in \mathcal{E}\}$,
$map_{R_2,R_1} = map^{-1}_{R_1,R_2}$,where $map^{-1} = \{(b,a)|(a,b) \in map\}$.

The set of attributes in the projection part of view V_{R_1,R_2} is $aset_{R_1,R_2}$ and for V_{R_2,R_1} it is $aset_{R_2,R_1}$.

Theorem 6 *The views V_{R_1,R_2} and V_{R_2,R_1} cooperate if the following holds:*
$map_{R_2,R_1}(map_{R_1,R_2}(aset_{R_2,R_1})) = aset_{R_2,R_1} \wedge$
$map_{R_1,R_2}(map_{R_2,R_1}(aset_{R_1,R_2})) = aset_{R_1,R_2}$.

If we consider the current case, the mapping $map_{R_2,R_1} = map^{-1}_{R_1,R_2}$ which means, that both mappings are in symmetry and Theorem 6 holds. The same is valid in the rule for *two entity/relationship types with nested tuple attributes*.

– **For two relationship types R_1, R_2 and semantically related attributes are not contained within the matching relationship type** have mappings the form:
$map_{R_1,R_2} = \{(a,b)|a \in aset^{e\_attr}_{R_2}, b \in R_2, (a,b) \in \mathcal{E}\}$,
$map_{R_2,R_1} = \{(a,b)|a \in aset^{e\_attr}_{R_1}, b \in R_1, (a,b) \in \mathcal{E}\}$.
The view cooperation must not necessarily take place because the mappings are not symmetric in general. The same conditions for the cooperation of views apply as in the Theorem 6.
We determine the set of view components (attributes) upon the both views V_{R_1,R_2} and V_{R_2,R_1} cooperate: $aset^{coop}_{R_1} = map_{R_2,R_1}(map_{R_1,R_2}(aset_{R_2,R_1}))$, $aset^{coop}_{R_2} = map_{R_1,R_2}(map_{R_2,R_1}(aset_{R_1,R_2}))$.
We adjust the set of attributes in each view definition: $aset_{R_1} = aset^{coop}_{R_2}$ and $aset_{R_2} = aset^{coop}_{R_1}$
Finally we modify the original mappings:
$map_{R_1,R_2} = \{(a,b)|a \in aset_{R_2}, b \in aset_{R_1}, (a,b) \in \mathcal{E}\}$,
$map_{R_2,R_1} = \{(a,b)|a \in aset_{R_1}, b \in aset_{R_2}, (a,b) \in \mathcal{E}\}$.
The equation $map_{R_1,R_2} = (map_{R_2,R_1})^{-1}$ now holds.

2.5 Translation into SQL

The view definitions we gained through the integration process can be translated to the desired SQL statements that create the views in the underlying database. The SQL standards SQL1999 and SQL2003 contain the support of complex data types with operations nest and unnest, user defined casting functions between different data types, and other extensions that enable the interoperability. At least these features have to be supported by the underlying DBMS (for example, DB2 V.8, Oracle 10) to create the views with the supporting functionality, e.g. instance mappings.

The translation of query views can be realized on the majority of today's DBMS. Update views, however, can not be directly translated. The SQL standards are continually extending the area of updatable views, but it is still not sufficient.

More complex update views must be broken into a set of simple views (presented functions in Sec. 2.2 are an option) and updated as a sequence in a transaction manner. An extensive computational support might be necessary.

3 Conclusion

Cooperation can succeed where integration is unlikely. We have considered cooperation between databases represented with HERM schemas using views. They work as functions that map the types on one another. The advantage is, if there doesn't exist an update view we can still query the relevant information. The user accesses the view in the same way as he would access the entity/relationship type in his schema. Not all constraints and conflicts can be handled on schema level. The importance of instance mapping functions is vital. The views are generated by rules. The aim of these rules is their simplicity and low complexity. Yet the range of conflicts they solve is broad. The updatability of the views in this paper has been reduced to key constraints, but the others can be easily added [Ves03].

Despite the benefits mentioned above, our approach has to be extended. The rules were designed as building blocks for other rules, e.g. integration of two relationship types containing nested tuple attributes and nested set attributes. Inference rules can be constructed that allow derivation of such new rules. Strategies for the generation of SQL statements and execution plans especially for update views are necessary. Update strategies must be developed to address more complex constraints. These strategies can be defined as compositions of mapping functions.

References

[BCVB01] Sonia Bergamaschi, Silvana Castano, Maurizio Vincini, and Domenico Beneventano. Semantic integration of heterogeneous information sources. *Data and Knowledge Engineering*, 36(3):215–249, 2001.

[CCG+03] Andrea Cali, Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, Paolo Naggar, and Fabio Vernacotola. Ibis: Semantic data integration at work. *Lecture Notes in Computer Science*, 2681:79–94, 2003.

[Con86] B. Convent. Unsolvable problems related to the view integration approach. *Proc. of International Conference on Database Theory*, pages 141–156, September 1986.

[GBMS99] Cheng Hian Goh, Stephane Bressan, Stuart E. Madnick, and Michael D. Siegel. Context interchange: New features and formalisms for the intelligent integration of information. *ACM Trans. on Information Systems*, 17(3):270–293, 1999.

[Hal01] Alon Y. Halevy. Answering queries using views. *Very Large Databases*, 10(4):270–294, 2001.

[Len02] Maurizio Lenzerini. Data integration: A theoretical perspective. In *21st ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems*, pages 233-246, 2002.

[Lev00] A. Levy. Logic-based techniques in data integration. In J. Minker, editor, *Logic Based Artificial Intelligence*. Kluwer Academic Publishers, 2000.

[PA03] P.McBrien and A.Poulovassilis. Data integration by bi-directional schema transfor-

[SP94] Stefano Spaccapietra and Christine Parent. View integration: A step forward in solving structural conflicts. *IEEE Transactions on Data and Knowledge Engineering*, 6(2):258–274, 1994.

[Tha00] B. Thalheim. *Entity-relationship modelling - Fundamentals of database technology.* Springer, Berlin, 2000.

[Tha01] B. Thalheim. Abstraction layers in database structuring: The star, snowflake and hierarchical structuring. Technical Report 13-01, BTU, Cottbus, 2001.

[Ves03] Vojtech Vestenický. Schema integration as view cooperation. Technical Report I-12-2003, BTU, Cottbus, 2003.

[VLF00] V. Vestenický, J. Lewerenz, and T. Feyer. Modelling the modification component of an information service. *Proc. of Challenges of the Symposium on Advances in Databases and Information Systems - ADBIS-DASFAA'2000*, (ISBN 80-85863-56-1):195–204, September 2000.

SCIENTIFIC PAPERS UNIVERSITY OF LATVIA. 2004. VOLUME 672, Datbases and Information Systems
62–77 P.

Optimization of continuous queries by regular inference

Dirk Kukulenz

Universität zu Lübeck,
Institut für Informationssysteme,
Ratzeburger Allee 160,
23538 Lübeck, Germany
kukulenz@ifis.uni-luebeck.de

Abstract. The information in the World Wide Web is subjected to a fast rate of change. Continuous queries, i.e. queries that concern a sequence of states of an information source in the future, are a means to capture these dynamics. 'Timer-based' continuous queries which are executed periodically usually have the disadvantage to be inefficient, information may get lost and the results may be nondeterministic. These problems are addressed in this article. We consider a specific pattern of information change in time, namely a change that may be described by a specific kind of a regular grammar. Examples for such time behavior may be found in web pages containing stock prices or satellite data. We present a procedure to estimate the parameters of the time behavior of such Web data based on regular grammar inference. It is shown how this estimation may be applied to acquire up to date information and thus to optimize continuous queries with a minimal number of network connections.
Keywords: dynamic web, continuous queries, regular inference

1 Introduction

The amount of information in the World Wide Web is increasing very fast. On the other hand much information is lost every day because it is deleted or replaced by new information. Problems resulting from this loss of information are inconsistencies, hyperlinks may e.g. point to data sources that no longer exist. Search engines may contain stale information and users usually don't have access to deleted information. A user applying a search engine has only access to a (frequently obsolete) screen shot of the Web, but not to the dynamic, time dependent characteristics of the Web.
One method to capture the dynamics of the Web is provided by *continuous queries* (c.q.). C.q. are similar to conventional queries to a data source, except that they are issued once and henceforth run 'continually' over the data source. Two main types of c.q. may be distinguished: change-based and timer-based continuous queries. Change based c.q., e.g. database triggers, are evaluated if a predefined event in the data source (e.g. the insertion of a new entry in a specific table) occurs, which then causes a user-defined action. In contrast to

this, timer-based c.q. are evaluated periodically, depending on the settings in the query. The problem concerning the realization of change-based c.q. in the Web, is how the query engine may be informed about the change of a remote data source. In this case, data source and query engine are usually separated and therefore change-based c.q. are difficult to be realized in a Web context. Timer-based c.q. have several disadvantages:

- *Nondeterministic results*: The records selected by a query depend on the time when the query is executed. Two queries executed e.g. every hour but with a different phase may lead to different results, because of a change in the data source between the query executions.
- *Inefficiency*: The execution of a query only makes sense, if the data source changed since the previous execution. If the query is executed too often, it may be inadequately expensive. Otherwise however, if the query is not executed frequently enough, information may get lost.
- *Age*: The query result may be stale for the period of time between a change in the data source and the execution of the query. If a query result is an information source for other users, e.g. a search engine index, these users may only acquire stale information.

This article addresses the problem of the optimization of timer-based continuous queries. The main issue for the solution is the estimation of the points in time where remote data-sources change. This information should however be acquired at minimal costs with respect to load on Web servers and Web traffic and the complexity of respective estimation algorithms. Different strategies are conceivable to detect and to predict the times at which data objects in the Web change. The simplest method in order to detect updates is to reload data objects frequently and to compare new and previously loaded data. The more frequent data sources are reloaded, the more exact the times of data changes may be estimated, but the more extensive is the network load. The HTTP-Protocol [4] provides different headers to support coherence mechanisms. The 'Last-Modified'-date is usually returned with a page returned from a 'Get' request. An 'Expires-Date'-header provided by authors or Web servers contains the time until the document will not be changed. Different strategies were developed to acquire optimal cache behavior based on http-headers [5],[13].
A problem with http-headers is that they are frequently not available. The 'Expires-Date' expects an author to estimate a document's lifetime, which is very difficult at the time it is created. Other headers also may not be available depending on the configuration of a respective Web server. A second problem is that a 'last-modified'-header doesn't contain information about the kind of a document's change. Many changes may not be relevant for a user since the content of a page isn't changed [11]. In this article we assume that none of the above http-headers are available. Documents are loaded as a whole and compared to each other.
The continuous queries addressed in this article only concern single objects in the Web (e.g. Web pages). The (similar) problem of optimizing the update of local copies of remote Web resources in the case of a large number of Web

pages, which is addressed e.g. in [3] or [2], has a different focus.
The update behavior of Web data in time may show different characteristics. A frequently applied model for update behavior are renewal processes [14]. Lifetimes of documents are supposed to be independent and identically distributed, e.g. by an exponential distribution [1]. In [3] page-changes are modeled by Poisson processes. One problem with these statistical models is the assumption of an independent and identical distribution of consecutive time intervals. Many examples are however conceivable where data are updated according to a well-defined pattern, e.g. every working day, not at night and not at weekends (figure 1). In this article we present an estimation method for similar update patterns.

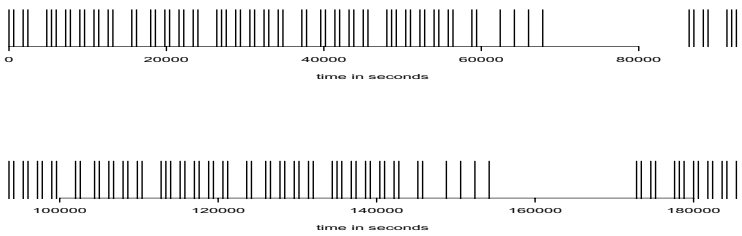


Fig. 1. The regular update behavior of the Web page 'http://www.ghcc.msfc.nasa.gov/GOES/goeswestpacus.html'. The pattern reappears every 86400 seconds ($\sim$ one day).

The set of the sizes of time intervals between updates has to be small compared to the total number of intervals and defines an alphabet. The sequence of time intervals defines a special case of a regular grammar we will denote as a *cyclic regular grammar*. Regular grammar inference is a problem well-known from machine learning [12], [6]. In [7] it is shown that it is not possible to exactly identify a regular language, given only positive presentations. Many algorithms were presented to learn regular grammars from positive and negative examples [8], [9], [10].
The cyclic-regular case considered here, which is defined in detail in section 2, is simpler than the general regular inference problem. In section 3 we propose an algorithm for cyclic-regular inference. In section 4 we show how this knowledge of regular behavior may be applied to find optimal reload times. The two algorithms are illustrated by examples in section 5. In section 6 we present experiments and in section 7 a summary is given and further aspects are discussed.

2 The model

Let $u_{p,i} \in \mathbb{R}^+$ denote the points in time at which the i^{th} update of page p occurs, where $0 \leq u_{p,1} \leq u_{p,2} \leq u_{p,3} \leq u_{p,4} \leq \ldots u_{p,n} \leq T \in \mathbb{R}^+, n \in \mathbb{N}$. We omit the page identification p in the following because we consider only one Web page at a time, i.e. $u_i := u_{p,i}$. The interval of time between the $i-1^{st}$ and i^{th} update will be denoted by $t_i := u_i - u_{i-1}, i \in \mathbb{N}$. Let $a_1, a_2, \ldots a_m \in \mathbb{R}^+$ denote the points in time where a continuous query engine requests the considered Web data object, where $0 \leq a_1 \leq a_2 \leq a_3 \ldots \leq a_m \leq T$. For $t \in \mathbb{R}^+$ let $N^u(t)$ denote the largest index of an element in the sequence u that is smaller than t, i.e. $N^u(t) := \max\{n | u_n \leq t\}$. Let $A^u(t) \in \mathbb{R}^+$ denote the size of the time interval since the last update, i.e. $A^u(t) := t - u_{N^u(t)}$. If t is the time of a query execution ($t = a_i$ for $i \leq m$), we denote $A^u(t)$ as the *age* of a_i. The age of a query execution denotes how much time since the last remote data update has passed and thus how long an old query result was stored although a new result should have been considered.[1]
Let $Q := \{t_j | j \leq n \in \mathbb{N}\}$ denote the set of time intervals between updates. We assign a symbol $s_i, i \in \mathbb{N}_{\leq n}$ to every element of Q. We call the set of symbols $\Delta := \{s_i | i \leq n\}$ the *alphabet* of the sequence u.
Let S denote a starting symbol, let $r_1, r_2, \ldots r_n$ denote terminals and the symbols $R_1, R_2, \ldots R_n$ non-terminals. In the following we refer to a regular grammar Γ of the shape:

$$
\begin{aligned}
S &\longrightarrow r_1 | r_1 R_2 | r_2 | r_2 R_3 \ldots | r_{n-1} | r_{n-1} R_n | r_n | r_n R_1 | \\
R_1 &\longrightarrow r_1 | r_1 R_2 \\
&\vdots \\
R_{n-1} &\longrightarrow r_{n-1} | r_{n-1} R_n \\
R_n &\longrightarrow r_n | r_n R_1
\end{aligned}
$$

as a *cyclic regular grammar*. The corresponding automaton is a non-deterministic

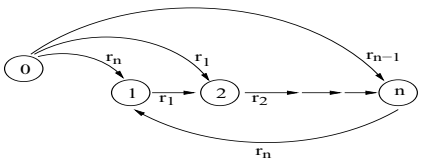


Fig. 2. Nondeterministic automaton corresponding to the grammar $(r_1 r_2 \ldots r_n)^\circ$. '0' is the starting state; every state except '0' is an accepting state.

[1] If the query result is used as an information source for users, in the respective period of time these users receive the old query result instead of the new one.

finite automaton (figure 2). To abbreviate this definition we will use the notation: $(r_1 r_2 ... r_n)^\circ := \Gamma$.
The first problem in the following is to describe the sequence of symbols $s_1, s_2 ...$ by a cyclic regular grammar of minimal size. The second problem is to predict further states of the automaton and to find optimal query execution times. Finding the optimum means that:

1. after each update of the remote data source, a query execution should be performed;
2. the query execution should follow the remote update as soon as possible, i.e. the sum of ages $\sum_{i=1}^{m} A^u(a_i)$ has to be minimal;
3. the number of query executions should be as small as possible.

One problem is that due to the discussion in the introduction the exact values for the update times are not known and have to be estimated by file comparison. Moreover, it is not possible to track data back in time.

3 Estimation of time-regular update behavior

3.1 Definitions

The presented algorithm for the estimation of cyclic-regular grammars consists mainly of two parts. One part is responsible for the estimation of symbols. The corresponding interval length of a symbol may not be determined exactly due to the fix size of the sampling interval and due to the fact that the points in time are real numbers. Therefore a maximal and a minimal length estimation value have to be provided in the symbol definition. A *symbol* is a 3-tuple $s = (i, max, min)$ consisting of a unique identifier i and two length parameters $s.max$ and $s.min$. A second component of the algorithm is responsible for the generation and the rejection of grammar hypotheses. A *hypothesis* $H = (\Gamma, s)$ is a 2-tuple consisting of a cyclic-regular grammar Γ and the current state s of the associated finite automaton, according to the enumeration of states in figure 2. In every step of the algorithm (i.e. after the detection of a symbol) the *default hypothesis* is added to the set of hypotheses. Taking the sequence of symbols registered by the system so far $(r_{i_1}, r_{i_2}, \ldots r_{i_p})$, the *default-hypothesis* is the cyclic regular grammar $(r_{i_1}, r_{i_2}, \ldots r_{i_p})^\circ$ with the corresponding automaton being in the state '1' according the enumeration of states in figure 2. This automaton accepts the sequence of input symbols. The last symbol is accepted by a transition from the last state to the first state. A *prediction* of a hypothesis H ($H.predict$) which is not in the start state (0) is the symbol, generated by a transition to the (unique) state following the current state. A *proceed* operation applied to a hypothesis H ($H.proceed$) which is not in the start state converts the current state of H into the subsequent state.

Grammar-Estimation

```
1       set starttime; set maxtime; set sampsize
        set last=0
        set min = 0;
        set symbols := ∅; set hset := ∅
2       load source
3       set previoustime=starttime
        set time=starttime+sampsize
4       wait until (time); load source
5       while time < maxtime
6         if N^u(previoustime) ≠ N^u(time)
7           if last ≠ 0
8             max = time-last
9             sym = Symbol-Assignment(max, min,
                        sampsize,symbols)
10            Update-Hypotheses(sym, hset)
11            set min = 0
12          set last = previoustime
          else
13          set min := min+sampsize
14        set previoustime:=time;
          set time :=time+sampsize
15        wait until(time); load source
16      output hset
```

Fig. 3. Main algorithm component for the estimation of a cyclic regular grammar.

3.2 Grammar estimation

Figure 3 shows the main algorithm component. This algorithm organizes the detection of symbols by requests to the remote server and it develops grammar hypotheses consistent with the sequence of registered symbols. In step 1 several values for variables are fixed. The starting time is set to be the current time. The *maxtime* variable is set to be the time supposed to be needed for the learning process, it has to be greater than the (estimated) sum of intervals of one cycle of symbols.[2] The *sampsize* value is the time between reloads (query executions) (we consider $a_j - a_{j-1}$ to be identical for each j). The value has to be significantly smaller than the expected time between different updates and the difference of update interval lengths. In step 3 the values for the time and the previous time value are fixed. The time value denotes the time of a reload (step 4). It is increased in step 14 until the maximal time (step 5) is reached. After every request it is checked in step 6, if the respective data object has changed since the last request. If a previous change in the data source has

[2] Many patterns will reappear every day or every week which is then a lower limit for the maxtime value.

been detected (stored by the variable 'last'), in step 8 the maximal length of the current time interval between two remote update operations is estimated to be the time between the current request and the time of the request before the previous detection of a change in the data source (max value in figure 5). Based on the *max* and the *min* value in step 9 the *Symbol-Assignment* component is applied to determine if a new symbol has been found or if an existing symbol has to be adjusted (section 3.3). This knowledge may be used to update the grammar hypotheses in step 10 applying the *Update-Hypotheses* component (section 3.4). After this step the temporary variables for the interval parameters are re-initialized in step 11. The time of the last detection of a data update is set to be the previous time in step 12 (figure 5). If by the current reload a data change hasn't been detected, the estimation of the minimal interval length is updated in step 13. When the *maxtime*-value is reached, the variable *hset* contains the grammar hypotheses consistent with the input sequence (step 16).

3.3 Symbol estimation component

```
Symbol-Assignment( max, min, sampsize, symbols)

1       for each symbol s in symbols:
2         if |max - s.max| ≤ sampsize
           and |min - s.min| ≤sampsize
3           set s.max = maximum{ s.max , max}
            set s.min = minimum{ s.min , min}
            return s
4       define new symbol sn
        set sn.max = max
        set sn.min = min
        add sn to symbols
        return sn
```

Fig. 4. Algorithm component for insertion of new symbols or adjustment of existing symbols.

Figure 4 shows the algorithm component *Symbol-Assignment*, responsible for the learning of new symbols or the adjusting of previously registered symbols, which is applied in step 9 of the main (grammar estimation) component. It depends on the variables *max*, *min*, the sampling-interval length *sampsize* and the current list of symbols, provided by the main component. The first two variables (*max* and *min*) imply an estimation for the maximal and the minimal interval length of a new observed interval between updates (figure 5). The algorithm decides, if a new symbol has to be inserted, or if a previously registered symbol has to be adjusted. In step 1 and 2 for each symbol in the set

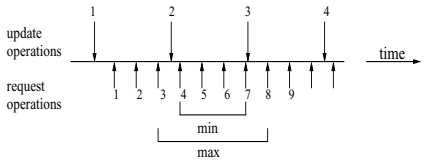


Fig. 5. Determination of a maximal and a minimal estimate for an interval length.

of symbols it is tested whether the new parameters are 'significantly different' with respect to the sampling size. If this is not true for one symbol, i.e. if the current symbol has already been detected, the parameters of this symbol are adjusted in step 3. It is the aim to minimize the *max* value and to maximize the *min* value in order to estimate an interval length or to localize an update as precise as possible (figure 5). This adjustment has to be incremental since it is not possible to track data back in time. If e.g. in figure 5 the 4th request would be just before the 2nd update, the *min* value would be 2 (times the sampling interval length), instead of a value of 3 in the figure. The actual values may therefore depend on the phase of the reload-request time sequence $(a_j)_j$. If the new interval-parameters are significantly different from all symbols defined so far, a new symbol is inserted into the set *symbols* in step 4. The algorithm returns a pointer to the new symbol or to the adjusted old symbol.

3.4 Grammar hypotheses organization

Figure 6 shows the algorithm component for the creation and the rejection of grammar hypotheses. After a symbol has been registered by the *Symbol-Assignment-* component the whole sequence of symbols observed so far is used to create the *default-hypothesis* (as defined in section 3.1) in step 1 of the algorithm. In steps 2 and 3 it is tested for each hypothesis H in the set of hypotheses if the prediction of H (3.1) corresponds to the newly observed symbol. If not,

Update-Hypotheses(newsymbol, hset)

```
1       add the default-hypothesis to hset.
2       for each H ∈ hset do
3         if newsymbol not equal to H.predict
4           delete H from hset
          else
5           apply H.proceed
```

Fig. 6. Update the set of hypotheses.

the hypothesis is deleted from the set of hypotheses in step 4. If the hypothesis is consistent with the observation the state of the hypothesis is increased in step 5. This algorithm guarantees that every hypothesis in the set of hypotheses is consistent with the sequence of symbols observed so far.

3.5 Limitations

In order to apply this algorithm, in particular the *Symbol-Assignment*-component, the variable *sampsize*, i.e. size of the time interval between requests $a_j - a_{j-1}$, has to be sufficiently small compared to the time between updates ($sampsize \ll \min_i t_i$). Moreover, the sampling size has to be smaller than the size difference between update intervals ($|t_i - t_j| > 2 \star sampsize$).

4 An algorithm for optimal data update

In this section we presume that the alphabet and the grammar have already been determined and we present an algorithm to find optimal reload times (figure 7). We assume that the current state of the automaton is not known. This state is determined in the first part of the algorithm in figure 7.
The structure of this phase detection is similar to the *Grammar-Estimation* algorithm. In step 1 the start time (the current time) and the size of the sampling interval are fixed, the data are loaded in step 2. In steps 3 and 4 the time value is increased and after the respective period of time, the data are reloaded. It is tested in step 5 if a unique state of the automaton (denoted also as *phase*) is available in the set $PhaseHypotheses$. In step 6 it is tested if a change in the data source occured. If previously, a change has been detected (step 7), the maximal size of the current time interval is adjusted (step 8). In step 9 the parameters of the current time interval are compared to the list of symbols of the alphabet and the respective symbol is determined. This symbol may correspond to more than one state of the automaton. These *phase hypotheses* are generated in step 10 and the respective hypotheses are subsequently deleted if not consistent with the sequence of input symbols in step 10 until a single state is left (example 2 in section 5). If by the current reload operation no change in the remote data source was detected, the estimation of the minimal current interval length (min) is adjusted in step 12. In step 13 the time value is increased, the source is reloaded and the algorithm continues in step 5.
After the detection of a unique phase, the respective hypothesis (automaton and state) is stored in the variable *Hypothesis* in step 14.
In the second part of the algorithm, optimal query execution times are determined by predicting the subsequent symbol.

Reload-Control (Input: estimated grammar)

```
1       set starttime; set sampsize; set last=0
        set max = 'large number', set min = 0

        // part1: determination of the phase
        set PhaseHypotheses := ∅
2       load source
3       set previoustime=starttime;
        set time=starttime+sampsize
4       wait until (time); load source
5       while |PhaseHypotheses| ≠ 1
6         if N^u(previoustime) ≠ N^u(time)
7           if last ≠ 0
8             max = time-last
9             Symbol sym = Symbol-Assignment(max,
                  min, sampsize, symbols)
10            create or adjust PhaseHypotheses
            else
11            set last = previoustime
          else
12          set min := min + sampsize
13        set previoustime:=time
          set time = time + sampsize
          wait until (time); load source
14      set Hypothesis as element in PhaseHypotheses

        // part2: continuous query exection
        set time = time - sampsize
15      while time < 'maximal c.q. time'
16        query execution
17        Symbol sym = Hypothesis.predict
18        Hypothesis.proceed
19        set forward = max {sym.min-sampsize,
                    sampsize }
20        set time = time + forward
          wait until (time)
```

Fig. 7. Algorithm to determine optimal reload times based on the grammar estimation in section 3.

The respective loop in step 15 is executed until the execution time specified in the continuous query is reached. The state of the automaton is increased (cyclically) in step 18. Since we know the subsequent state (step 17), we also have an estimation for the respective size of the time interval. This knowledge is used to compute an optimal *forward* time in step 19. In step 20 the time value is increased.

5 Examples

Example 1:
In order to illustrate the *Grammar-Estimation* algorithm we assume that the system registers remote updates as depicted in figure 8. Impulses of length 1 denote reload requests, impulses of length 2 denote registered changes in the remote data source in step 6 of the *Grammar-Estimation* algorithm.

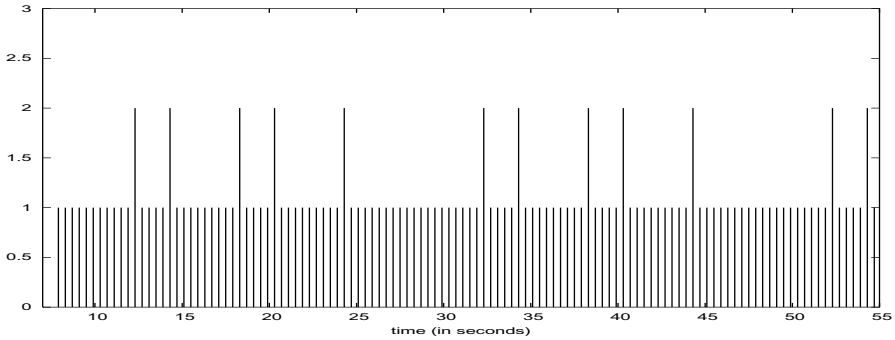


Fig. 8. Update times as registered by the *Grammar-Estimation*-algorithm. Impulses of length 1 denote reload operations. Impulses of length 2 denote observed changes in the remote data source.

According to figure 8, the sequence of detected symbols is *ababcababca* in this chronological order (if the respective intervals are denoted as *a, b* and *c*). After detecting the symbol *a*, the default hypothesis $H1 := (a)^\circ$ is inserted into the empty set *hset* (table 1) in step 10 of the *Grammar-estimation* algorithm. This hypothesis is in the first state. In step 2 in table 1 the second detected symbol is *b* which is different to the prediction of $H1$ ($H1.prectict = a$). Therefore $H1$ is deleted from *hset*. Again, the default hypothesis $H2 := (ab)^\circ$ is added to *hset*. In step 3 the symbol *a* is detected. In this case the prediction of $H2$ is true. Therefore the state of $H2$ is increased. The default hypothesis $H3 := (aba)^\circ$ is added to hset. This procedure continues until in step 5 the symbol *c* is detected. This hypothesis is the first one to be consistent with the sequence *ababcabab* and it also turns out to be the smallest one.

Example 2:
In a second example we want to illustrate the *Reload-Control*-algorithm. We assume that the estimated grammar is
$(ababab)^\circ$ (different from the grammar above). For the phase detection we assume that the automaton is in state 2 (according to figure 2). This state is unknown to the algorithm and has to be determined by the first component of the *Reload-Control* algorithm. The first detected symbol is *b* (table 2). The set of phase hypotheses (*PhaseHypotheses*) is empty in step 5 of the *Reload-*

step	input symbol	reject hypothesis	insert hypothesis	hypotheses/state
1	a		H1:=$(a)^\circ$	H1/state=1
2	b	H1	H2:=$(ab)^\circ$	H2/state=1
3	a		H3:=$(aba)^\circ$	H2/state=2 H3/state=1
4	b	H3	H4:=$(abab)^\circ$	H2/state=1 H4/state=1
5	c	H2,H4	H5:=$(ababc)^\circ$	H5/state=1
⋮				

Table 1. Computation steps of the *Grammar-Estimation*-algorithm for the sequence *ababc*....

Input symbol	delete phase	phase hypotheses
b		3,5,7
a	7	4,6
b		5,7
a	7	5

Table 2. Detection of the phase in the *Reload-Control*-algorithm for the grammar $(ababab c)^\circ$, starting in state 2.

Control-algorithm. Therefore the set of phase hypotheses is created in step 10. The possible states of the automaton are 3, 5 and 7 (table 2). After detecting the symbol a in the next step the phase hypothesis 7 is deleted since the prediction would be the symbol c. The states of the other hypotheses are increased. The third detected symbol b is consistent with both remaining hypotheses and the states are again increased. After detecting symbol a, the hypothesis 7 has to be deleted and the hypothesis 5 is the single phase hypothesis left. At this time the algorithm knows the state of the automaton and may therefore predict further states.

6 Experiments

In order to evaluate the *Grammar-Estimation*-algorithm it has to be tested if the estimated grammar corresponds to the original grammar of the remote update times. Usually this grammar is not known since the source code of a respective Web service is not known. In order to test the grammar estimation it is easier to assume a certain grammar, to generate data corresponding to this grammar, i.e. to modify a data source at the specific points in time, and to apply the algorithm to this data source. Now the original grammar and the estimation result may easily be compared.
We applied this procedure for two different grammars. For both grammars, the constraints at the end of section 3 are fulfilled. The first considered grammar is $(abbc)^\circ$. The symbol 'a' denotes a time interval of a length of 2 seconds, 'b' and 'c' denote time intervals of 4 and 8 seconds. The sampling interval in the experi-

ment is 0.4 seconds. The *starttime*-value in the *Grammar-Estimation* algorithm was set to be 7.5 and the *maxtime*-value was set to be 60 seconds. Figure 9 shows the output of the algorithm (step 16 in figure 3). Three symbols (s0, s1 and s2) are detected with an error of the interval length of approximately twice the length of the sampling interval. Two grammar hypotheses are consistent with the input data. The smallest hypothesis, which is the hypothesis of choice, is the input grammar.[3]

```
symbol      min    max
0           7.600  8.399
1           1.6    2.399
2           3.599  4.399
list of consistent hypotheses
0 1 2 2
0 1 2 2 0 1 2 2
```

Fig. 9. Output of the *Grammar-Estimation* algorithm for the automaton $(abbc)^\circ$ (text).

```
symbol      min     max
0           1.6     2.400
1           3. 599  4.400
list of consistent hypotheses
0 1 1 1 1
0 1 1 1 1 0 1 1 1 1
0 1 1 1 1 0 1 1 1 1 0 1 1 1 1
```

Fig. 10. Output of the *Grammar-Estimation* algorithm for the grammar $(aaaab)^\circ$.

Figure 10 shows the output of the algorithm for the grammar $(aaaab)^\circ$. A similar grammar may occur if a data source is updated every working day but not on Saturdays and Sundays.
Based on the previous grammar estimation we may now apply the reload-control algorithm to find optimal update times. Figure 11 shows the result for the grammar $(abbc)^\circ$. In the first phase of the algorithm (until time$\approx$20 sec.) the algorithm determines the state of the automaton. After this phase detection, the algorithm may predict further symbols and thus compute optimal reload times. A few reload requests are pursued before an update[4] and one request is performed directly after the update. The number of reloads is thus minimized

[3] Due to the definition of a *cyclic regular grammar* $(abbc)^\circ = (cabb)^\circ$.
[4] This is necessary because of the error in the estimated interval lengths.

and the local data are updated directly after the change of a data source in order to keep the age minimal.
The age of the query result as defined in section 2 is related to the size of the sampling interval. We may expect that the smaller the sampling interval, the smaller is the (average) age of query results.

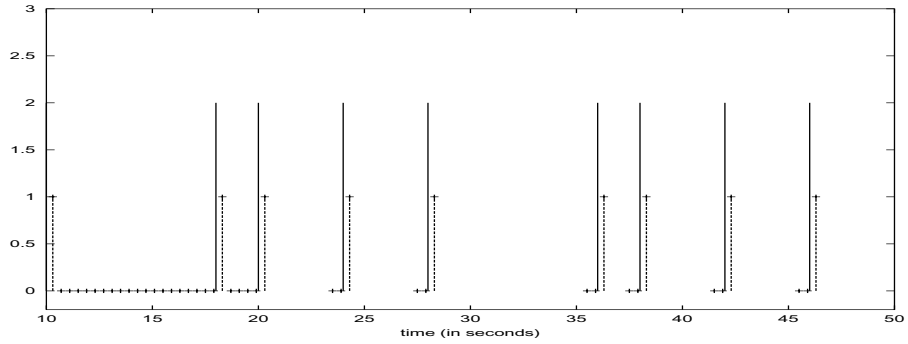


Fig. 11. Application of update prediction according to algorithm 7. The impulses of length 2 denote remote updates. The '+' symbols denote reload requests. Impulses of length 1 denote observed updates.

In figure 12 an experiment is shown to prove this assumption. In the experiment the size of the sampling interval was increased from 0.05 to 0.41 seconds. The age of the query results was determined for a specific amount of time (60 seconds). The figure shows the mean values and the variances of the results for different legths of the sampling interval. In fact, the age increases when the sampling size increases. However due to the fact that an update interval may be a multiple of the sampling interval, the graph is not a straight line. The figure also shows the number of reload requests for different sampling intervals. The $\times$-symbols denote values depending on the number of requests needed in the grammar estimation process: $(1 - 1/(\text{number of reloads}))$. In order to optimize continuous queries it is necessary to minimize both, the number of queries and the age. Figure 12 makes clear that both requirements are contradictory and an optimal choice depends on a user's priorities.

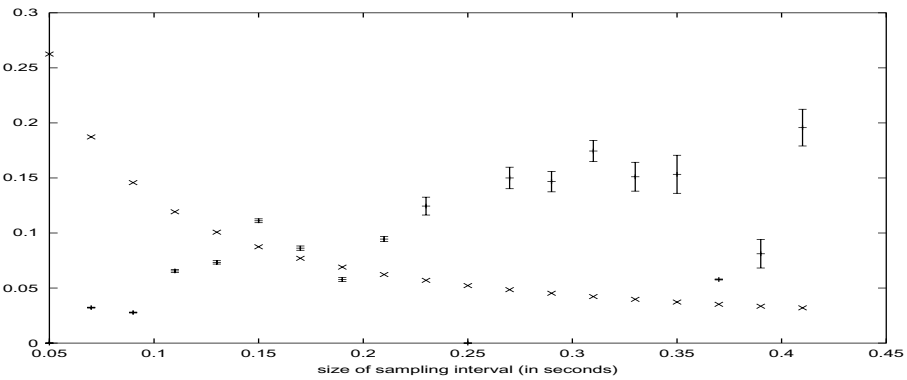


Fig. 12. The figure shows the connection between the size of the sampling interval (x-axis) and the age of query results (bars) and the number of necessary reloads needed in the grammar estimation process (×). The error-bars (mean and variance) denote the age of data for different sampling intervals. The ×-values are related to the number of requests in the grammar estimation process (text).

7 Conclusion

Continuous queries are a means to capture the dynamics of data on the Web. Change-based c.q. are in most cases not adequate in a Web context because remote changes of data sources are usually not known on the client side. The optimization of timer-based c.q. makes the estimation of the behavior in time of remote data sources necessary. The update behavior may follow different rules and diverse (statistical) models may be applied in this context. Other research was concerned with models like Poisson processes or renewal processes. One main disadvantage of these methods is that successive time intervals are usually regarded to be independent. However many examples exist, where this assumption is not true. In this article we present an algorithm to estimate the parameters of an update behavior of data objects that may be described by a specific kind of a regular grammar. This algorithm takes into account that the points in time where data sources in the Web change may usually not be registered exactly. The estimated grammar is used to develop an algorithm to determine optimal points in time for query executions. 'Optimal' means that every update of the original source should be observed and followed by a query execution as soon as possible. The number of query executions should be as small as possible in order to minimize network traffic and server load. These requirements are fulfilled by the algorithm as shown in the experiments. In order to optimize c.q., up to now the algorithms may only be applied to Web data like stock and satellite data. In order to apply this method to a larger range of Web data, the method has to consider further statistical properties of Web data. The remote update times may follow a certain distribution (the Web page of a newspaper may e.g. be updated every day except Sundays between

8h and 12h am). For this problem regular and statistical properties have to be combined. In the learning phase, some update values may be missing. The algorithm should nevertheless be able to find an optimal grammar in the case of distorted input data.

References

1. B. E. Brewington and G. Cybenko. How dynamic is the Web? *Computer Networks (Amsterdam, Netherlands: 1999)*, 33(1–6):257–276, 2000.
2. J. Cho and A. Ntoulas. Effective change detection using sampling. Technical report, UCLA Computer Science Department, 2002.
3. E. Coffman, Z.Liu, and R.R.Weber. Optimal robot scheduling for web search engines. *Journal of Scheduling*, 1(1):15–29, June 1998.
4. W. W. W. Consortium. W3c httpd. http://www.w3.org/Protocols/.
5. A. Dingle and T.Partl. Web cache coherence. *Computer Networks and ISDN Systems*, 28(7-11):907–920, May 1996.
6. P. Dupont, L. Miclet, and E. Vidal. What is the search space of the regular inference? In R. C. Carrasco and J. Oncina, editors, *Proceedings of the Second International Colloquium on Grammatical Inference (ICGI-94): Grammatical Inference and Applications*, volume 862, pages 25–37, Berlin, 1994. Springer.
7. E. Gold. Language identification in the limit. *Information and Control*, 10:447–474, 1967.
8. J. Oncina and P.Garcia. Inferring regular languages in polynomial update time. *Pattern Recognition and Image Analysis, Perez, Sanfeliu, Vidal (eds.), World Scientific*, pages 49–61, 1992.
9. R. Parekh, C.Nichitiu, and V.Honavar. A polynomial time incremental algorithm for learning dfa. In *Grammatical Inference: 4th International Colloquium, ICGI-98, Ames, Iowa, USA*, volume 1433, page 37, 1998.
10. R. Parekh and V. Honavar. Learning dfa from simple examples. *Machine Learning*, 44(1/2):9–35, 2001.
11. S. C. Rhea, K.Liang, and E.Brewer. Value-based web caching. In *WWW 2003*, pages 619–628, 2003.
12. S. Shapiro, editor. *Encyclopedia of Artificial Intelligence*, chapter Inductive inference., pages 409–418. John Wiley and Sons Inc., 1987.
13. D. Wessels. Intelligent caching for world-wide web objects. In *Proceedings of INET-95, Honolulu, Hawaii, USA*, 1995.
14. J. L. Wolf, M. S. Squillante, P. S. Yu, J. Sethuraman, and L. Ozsen. Optimal crawling strategies for web search engines. In *Proceedings of the eleventh international conference on World Wide Web*, pages 136–147. ACM Press, 2002.

SCIENTIFIC PAPERS UNIVERSITY OF LATVIA. 2004. VOLUME 672, Datbases and Information Systems
78–92 P.

Competition-based Query Execution in Web Environment

Juliusz Jezierski, Tadeusz Morzy

Poznan University of Technology
Piotrowo 3a, 60-965 Poznan, Poland
{jjezierski, tmorzy}@cs.put.poznan.pl

Abstract In Web environment sites and network can exhibit widely fluctuating characteristics. Moreover, assumptions concerning the availability of data statistics and the predictability of delays in access to sites rarely hold in such environment. As a result, the traditional static cost-based query optimization approach based on data statistics cannot be directly applied to this environment. In this paper we propose a novel competition-based query optimization and execution strategy which is able to cope with the lack or limited availability of data statistics and unpredictable delays in access to data sources. The basic idea is to initiate simultaneously several alternative query execution plans and to measure dynamically their progress. Processing of the most promising plan is continued, whereas processing of remaining plans is stopped. We also present a performance study of our approach using workloads consisting of queries from the $TPC-H$ benchmark.

Keywords. Query execution, Web systems, unpredictable delays, unavailable statistics

1. Introduction

There is an increasing interest in query optimization and execution strategies for the Web environment that can cope with two specific properties of this environment: the lack or limited availability of data statistics and unpredictable delays in access to data sources. Typically, query processing parameters may change significantly over time or may be simply not available to query engines in the Web environment. Web sites that disseminate data in the Web environment in the form of files, dynamically generated documents, and data streams usually do not allow access to internal data statistics. The postulate to make local statistical information available to external applications is unrealistic due to the local autonomy of Web sites. Moreover, the idea of dedicated service construction to gather and maintain data statistics may be hard to accept and implement. First of the, a process of gathering and maintaining data statistics concerning external data sources is very expensive and may significantly affect local processing at data sources. Secondly, such process is limited to data

Source	size [kB]	Count[tuple]
A	800	$5 * 10^3$
B	1500	10^4
C	2000	$2 * 10^4$

Table 1. Volumes of sources (Example 1)

Join	selectivity
A⋈B	$2 * 10^{-3}$
A⋈C	$2 * 10^{-4}$
B⋈C	$2 * 10^{-5}$

Table 2. Selectivities of joins (Example 1)

whose localization is given by a well-known URL address. Many Web data are dynamically generated on the basis of detailed selection criteria specified by users. To obtain data statistics concerning these kinds of data it would be necessary to specify correct values of these criteria. This process requires human interaction and makes the automatic maintenance of statistics impossible.

The second specific property of the Web environment is the unexpected delay phenomenon in access to external data sources. Such delays may cause significant increase of the system response time. They appear due to variable load of network devices resulting from a varying activity of users, and also, due to breakdowns.

For all of these reasons, it is not appropriate to use traditional static cost-based query optimization and execution strategies in Web environment. In fact, specificity of data processing in the Web environment strongly limits or disqualifies most of classical static distributed query optimization techniques proposed in the literature and requires new, dedicated solutions. A key requirement of a query optimization and execution strategy for the Web environment is that it generates and processes query execution plans in an unpredictable and constantly fluctuating environment.

1.1. Motivating examples

The following examples illustrate the impact of limited availability of statistics and unpredictable delays on selection of the "optimal" query execution plan.

Example 1 Given data sources: A, B, C described by statistics depicted in Tables 1 and 2. Let us consider the following SQL query:

```
select * from A, B, C where A.b=B.a and B.c=C.b (Q1)
```

Let us assume that B⋈C join selectivity coefficient as well as sizes of data sources A, B, and C are available. Moreover, let us assume that the coefficient of the join selectivity of A⋈B is unknown. To evaluate the costs of potential execution plans of the query Q1 and to find the optimal plan using the traditional cost based query optimizer, we may assume two hypothetical values of A⋈B join selectivity coefficient: sel1(A⋈B)=0.01 and sel2(A⋈B)=0.1. For the former value of A⋈B join selectivity coefficient, the optimal plan would be P1(Q1)=(A⋈B)⋈C, and the cost of the plan is Cost(P1)= 1960. For the later variant of the A⋈B join selectivity coefficient, the cost of the plan P1 increases to 14200. The optimal plan, assuming the second variant the A⋈B join

Source	size [kB]	Count[tuple]
A	800	$5 * 10^3$
B	1500	10^4
C	2000	$2 * 10^4$
D	6300	$5 * 10^4$
E	9900	10^5

Table 3. Volumes of sources (Example 2)

Join	selectivity
A⋈B	10^{-2}
B⋈C	$1.2 * 10^{-2}$
B⋈D	$9 * 10^{-3}$
C⋈E	10^{-2}

Table 4. Selectivities of joins (Example 2)

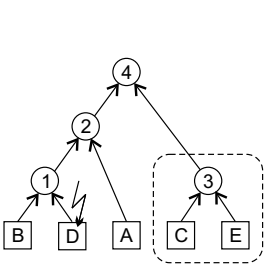


Figure 1. Plan P3

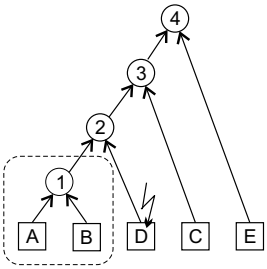


Figure 2. Plan P4

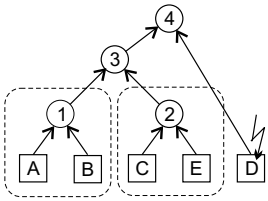


Figure 3. Plan P5

selectivity coefficient, would be P2(Q1)=(B⋈C)⋈A with the cost Cost(P2)= 10680.

Notice that, depending on the value of the A⋈B join selectivity coefficient, the cost of a query execution plan may vary significantly (e.g., P1 varies from 1960 to 14200). It is easy to notice that even the lack of knowledge about single statistic prevents from finding an optimal query execution plan for a query. In our approach we propose simultaneous execution of plans P1, P2, and monitoring their progress. Dynamically gathered statistics enable us to cancel a plan, which is more expensive. Finally, the query result will be produced by the plan which is the best in real environment conditions.

Example 2 Given data sources: A, B, C described by statistics depicted in Tables 3 and 4. Let us consider the following SQL query:

```
select * from A, B, C, D, E
where A.b=B.a and B.c=C.b and B.d=D.b and C.e=E.c (Q2)
```

Let us assume that all data statistics are available. Using the classical cost based query optimizer we can evaluate the optimal query execution plan for Q2, denoted P3: ((B⋈D)⋈A)⋈(C⋈E) (Figure 1). The cost of P3 is Cost(P3)= 5936. Let as further assume that the actual values of some statistics which were collected during the execution of P3 differ from the values assumed by the optimizer to calculate the optimal query execution plan for Q2, namely: sel(B⋈D)=0.011 and sel(C⋈E)=0.06. In this case, the cost of P3 is 30064. For these values of join selectivity coefficients another query execution plan P4 is optimal for Q2 (Figure 2): P4:(((A⋈B)⋈D)⋈C)⋈E, Cost(P4)= 29736. Query execution system might suspend the execution of and start the execution P4.

If unexpected suspension of download from the source D occurs during P4, then this plan will be blocked in the operator 2. Similarly, the plan P3 will also be blocked in operator 1. In our approach we propose to exploit partial results of the plans P3 and P4 to construct a new plan which can be continued despite the unavailability of the source D. In this case, it is possible to use the partial result of the operator 1 of the plan P4 and the partial result of the operator 3 of the plan P3. The query optimizer may use partial operators results since query engines usually use symmetric hash join operators to join data from Web sources [10, 6, 8, 2, 12], which materialize join arguments in hash tables. Therefore, we may use these hash tables to extract partial results of the query execution plan. Using partial results of different query execution plans it is possible to construct a new plan P5: $((A \bowtie B) \bowtie (C \bowtie E)) \bowtie D$ with Cost(P5)=40644 (Figure 3). This plan is more expensive than both P3 and P4, but because of unexpected delay of download data from the source D it provides better response time.

1.2. Related work

The query optimization problem in wide area network environment was intensively studied in two different fields of research: query plans reoptimization techniques [11, 7, 9] and adaptive join operators [13, 4, 10, 6, 8, 2, 12].

In [11] the authors propose the query reoptimization strategy that minimizes the system response time in case of unexpected delays in access to data sources. The idea is the following. The query engine starts with a query execution plan generated by the cost based query optimizer. The plan does not take into account dynamic changes of network parameters. Then, the plan is executed by the engine equipped with the delays detector. After detecting a delay, control is passed to the query optimizer, which dynamically reconstructs the query execution plan. The plan can be changed either by operator reschedule or operator synthesis. Operator rescheduling consists in changing operators' order in the plan. This technique does not change a general shape of the original plan. Operator synthesis consists in changing the original plan: some operators are removed from the plan; some are added to the plan. This kind of reoptimization is able to hide delays in access to data sources. The engine tries to do its best while waiting for delayed sources. The selection of an alternative plan is made in a way that minimizes the difference in the cost of original plan and potential alternative plan. In [7] authors present query reoptimization techniques that prevent execution of ineffective plans that could be generated due to unreliable data statistics and exponentially cumulated statistical errors. During query execution data statistics concerning intermediate query results are collected. If the difference between optimizer' estimated statistics and runtime statistics is too high and the profit following from generation of a new plan compensates the cost resulting from repeated optimization process, then the reoptimization is made. In [9] authors consider the unexpected delays problem caused by unknown times of local queries execution on local databases in multidatabase environment. To mask delays in access to data sources, they

propose the strategy which, during each iteration of query execution, considers only a set of data sources available in a given time point. From this set of sources, two sources that can be joined with the least cost are selected. In order to avoid extremely expensive joins a maximal accepted cost is defined. If a cost of a chosen operator is greater than the maximal accepted cost, then the algorithm waits for responses from other data sources. These sources can probably provide potentially less expensive plans to finish query execution.

The aim of the research on new join operators is to provide the successful join execution in case of delays in access to join arguments. In [13] the symmetric hash join operator is presented, which computes data from two sources. The operator uses two hash tables, one for each data source. In [4] authors propose a modification of the solution from [13] in order to give users statistically reliable partial results of aggregation (e.g., sum, average). In [10, 6, 8] the scalability of symmetric hash join operator problem in case of limited RAM memory is considered. The authors present alternative algorithms of swapping intermediate results to disk. In [2] a new multi-argument operator that integrates selection and join of data from many data sources is proposed. This approach uses dynamic route of data inside operator to modify order of operations accordingly to dynamic changes of parameters of data sources. In [12] a multi-argument join operator is proposed to join several data streams based on common join criterion. The authors concentrate on scalability of the proposed operator in case of limited RAM memory.

An extensive survey on dynamic query optimization techniques was published in [3].

The above mentioned results are very interesting and promising from the point of view of the query optimization problem in the Web environment. Some of these results are already partially implemented in commercial utilities. However, it is necessary to point out that proposed solutions do not solve the general query optimization problem for the Web environment with limited availability of statistics and unpredictable delays. Some proposals omit the problem of limited availability of statistics [11, 7, 9, 12], other solutions tackle the problem assuming strong restrictions concerning the query shape [2, 12], the choice of data access methods [2, 12], or scalability [2].

1.3. Contribution

In the paper, we present the novel competition-based strategy of query execution in the wide area network environment that solves or reduces the limitations of previous solutions. The basic idea behind the proposed strategy is the observation that due to limited availability of data statistics and unexpected delays in access to external data sources it is impossible to generate a single "optimal" query execution plan. The proposed competition query execution strategy consists in simultaneous execution of a set of alternative query execution plans for a given query. The system monitors the execution of these plans, and the most attractive plans are promoted, while the execution of the most expensive plans is canceled. The final query result is delivered to the user by the plan that

has won the competition according to the rules defined by the strategy implementation. Our competition strategy was inspired by the idea presented in [1]. The author presents theoretical and practical analysis of processing the selection operation in relational database systems in case of unknown or unreliable statistics. To select specified data, the author proposes simultaneous processing, according to well-defined criteria, of full scan and index scan operators. The solution considered in the paper limits the competition to individual data access operators. In our approach, competition concerns entire query execution plans or subplans and the proposed algorithm adapts the competition process to specificity of the wide area network environment.

The paper is organized as follows. In Section 2 we overview the proposed competition-based strategy of query execution. In Section 3 we present the greedy algorithm of query execution employing the competition-based strategy. The experimental evaluation in Section 4 demonstrates the performance of the strategy. The paper is concluded in Section 5.

2. Competition-based strategy of query execution

In traditional database systems the query specified by the user is transferred to the query optimizer, which chooses the optimal query execution plan (QEP) during query compilation. The query optimization process depends on: (1) the cost function used to evaluate the cost of a query, (2) the search space of all possible QEPs for a given query, and (3) the search strategy used to penetrate the search space of QEPs. The final QEP generated by the query optimizer is static and it does not change during its execution.

Due to specific properties of the wide area network environment, this traditional approach has to be revisited. Limited availability of data statistics makes generation of one optimal QEP impossible. Moreover, unexpected delays in access to data sources require periodical reorganization of QEP to minimize the impact of delays on the system response time.

Our query execution strategy is based on the idea that the query optimization process should be continuous and interactive, which means that the search space of QEPs should be also analyzed during query execution. The query optimizer improves the initial QEP by taking into account data statistics gathered during query execution. Figure 4 illustrates the basic idea of the approach.

The proposed strategy consists in simultaneous processing of many, incomparable QEPs and monitoring their progress. The results of QEPs competition are analyzed to stop and prune QEPs which are outperformed by other QEPs. During plans' competition, it is possible to start processing of new QEPs. Finally, we receive a single QEP as the result of the competition.

Formally, the competition-based strategy of query execution can be defined as a triple:

CSQE = {PGR, CC FR}, where:
PGR - rules of plan generation,
CC - competition criteria,

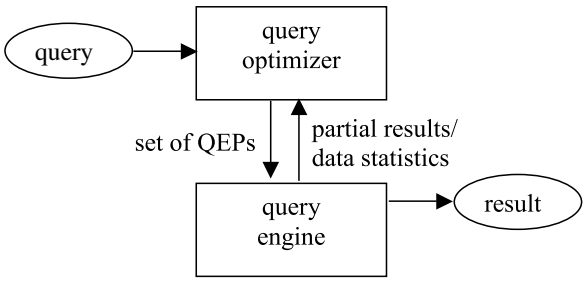


Figure 4. Architecture of the query optimization process

FR - feedback rules.

PGR denotes a set of rules used to generate QEPs participating in the competition. The important issue is the proper selection of initial QEPs. On the one hand, in order to reduce the overhead related to simultaneous processing of many QEPs, it is necessary to restrict the number of initiated plans. On the other hand, if the number of initiated plans is too small, then the adaptation to changing conditions of the runtime environment is automatically restricted. CC denotes the competition criterion used to evaluate the attractiveness of different QEPs. Proper definition of CC allows to limit the overhead related to simultaneous processing of many plans by pruning ineffective QEPs. FR denotes a set of rules that control the competition process (e.g., start new plans). FR allows to adapt query execution process accordingly to changes of runtime environment parameters, e.g., delays in access to data sources.

The competition-based strategy has an "open" character and it can be implemented in many different ways. To illustrate the approach, in the next section we present the greedy algorithm which implements the competition-based strategy of query execution.

3. Algorithm description

The algorithm is based on the producer-consumer model of processing and multithreading. We assume no availability of statistics. All necessary data are dynamically established or estimated during query execution. Below, we present the pseudocode of our algorithm (abbreviated as GA).

```
1: sourceSet=initiateAccessToAllSources();
2: while |sourceSet| > 1 do
3:    message=waitForMessage();
4:    if message.operation=='initiated' then
5:       Source=message.initiatedSource;
6:       competitors=constructAllSubplans(query, source);
7:       competGroup=findCompetGroup(source);
8:       if competGroup!=null then
9:          appendToCompetGroup(competGroup, competitors);
```

```
10:       else
11:         competGroup=new CompetGroup(competitors);
12:         runCompetGroup(competGroup);
13:       end if
14:     else if message.operation=='statistic' then
15:       competitor=message.competitor;
16:       if isWinner(competitor, competitor.competGroup then
17:         removeFromCompetGroupAllExcept(competitor);
18:       else if isLost(competitor, competitor.competGroup) then
19:         removeFromCompetGroup(competitor);
20:       end if
21:     else if message.operation=='finish' then
22:       sourceSet-= message.competitor.leftSource;
23:       sourceSet-= message.competitor.rightSource;
24:       removeCompetGroup(message.competGroup);
25:       sourceSet+= message.result;
26:       reconstructQuery(query);
27:       sendMessage(new Message('initiated', message.result));
28:     end if
29: end while
```

Input data for the algorithm are the following: the location of data sources, access methods to data, and the specification of a query. In the first step, the algorithm initiates access to data sources (line 1). Due to possible delays data are available from different sources at different time points. The algorithm continues as long as there are any data sources available (line 2).

The algorithm waits in a loop for messages (line 3). If a message was sent by a thread which waits for a source opening (line 4), then all two-argument subplans are constructed applying query join condition. The first argument is the newly opened source (line 6), and the second argument is a source which was opened earlier. Then, the algorithm generates a group of competitive subplans (line 7). A subplan is added to the group if at least one of its arguments is used by any other subplan already belonging to the group. The concept of a group allows to generate bushy plans and to reduce risk of omitting attractive subplans. The newly initiated subplan might belong to two groups. In this case, the algorithm integrates these two groups and the subplan is added to this integrated group. Due to the lack of space and in order to improve algorithm readability, we have omitted the description of the group integration process in the algorithm listing. If the algorithm does not find a proper group to which generated subplans should be added, then it creates a new group (line 11) and starts competition for subplans in this group (line 12). Steps from line 4 to line 12 implement PGR of competition-based strategy of query execution.

Processing of each subplan generates a message containing information about subplan's progress and about data statistics. Each subplan is implemented by the operator called Scalable Ripple Join [8]. This operator can estimate the subplan selectivity coefficient using cross sampling method [5] and can determinate statistical error of this selectivity. After collecting a sam-

ple, the operator generates a message containing the current value of estimated selectivity. Haas [5] proved convergence of cross sampling, so each subsequent message generated by the operator contains more precise estimation of the selectivity coefficient.

After receiving a message with statistics, which contains statistics generated by the operator (line 14), the algorithm tries to arbitrate the competition within a given competition group. If the selectivity of a given operator (i.e. of a given subplan) is higher than the selectivity of all remaining operators in this group (line 16), then this operator becomes the winner of the competition and remaining operators are canceled. If the selectivity of a given operator is lower than the selectivity of its competitors (line 18), then the operator is canceled. Steps from line 14 to line 20 implement CC of competition-based strategy of query execution.

After receiving a message informing about the termination of a subplan (line 21) the algorithm removes the sources of the subplan from the set of initiated sources (lines 22, 23). Moreover, the algorithm removes the corresponding competition group and cancels all participants of this competition. In the next steps, locally materialized result of the terminated operator is added to the set of initiated sources (line 25). Then, the query is reconstructed to take into consideration processing progress of the original query (line 26). Finally, the algorithm sends a message that informs about newly created source (line 27). Steps from line 21 to line 27 implement FR of competition strategy of query execution.

Processing is continued until all sources are downloaded and all join operations specified in query are finished.

Computational complexity of the greedy algorithm As the complexity yardstick we take the number of started subplans in the competition. If a given query references n sources then the greedy algorithm starts *n-1* times the competition. In the worst case, in each *k-th* competition participate $\frac{(n-k+1)(n-k)}{2}$ subplans. Hence, computational complexity of the greedy algorithm is $\sum_{k=1}^{n-1} \frac{(n-k+1)(n-k)}{2} = \mathcal{O}(n^3)$.

4. Experimental evaluation

In order to show the performance and practical relevance of the proposed competition query execution strategy, we have performed a series of experiments to evaluate our approach and compare it to an alternative solution. As we already mentioned before, none of the proposed solutions so far can cope with the general problem of query optimization in the Web environment with limited availability of data statistics and unpredictable delays in access to data sources. Therefore, to demonstrate the practical relevance and universality of our strategy, we compare it with a simple "brute-force" strategy (abbreviated as BFS), which generates all possible query execution plans of a given query, and to comparison we have taken into account the average value of their results.

Source	Tuples	Kbytes	Primary Key
Region	5	0.4	regionkey
Nation	25	2	nationkey
Supplier	10K	1,300	suppkey
Customer	150K	23,000	custkey
Order	1,500K	158,000	orderkey
Part	200K	23,000	partkey
Lineitem	6,000K	673,000	orderkey+linenumber
PartSupp	800K	111,000	suppkey+partkey

Table 5. TPC–H database schema and sizes

We considered two basic performance evaluation criteria: the system response time and the volume of processed data. We choose the second criterion to uniform present load of main computer resources: CPU, disk and network card. The main goal of the experiment is the analysis of the impact of three factors: transfer rates, initial delays, and allocated main memory, on the performance evaluation result.

Our experiments were performed using a query engine that implements our competition-based strategy. The workload based on queries from the TPC-H benchmark. The database is based on a TPC-H Scaling Factor (SF) of 1, and is described in Table 5. We have chosen two of the TPC-H queries (Q8 and Q9) for our experiments. These queries were chosen because they are fairly complex (6 to 8-way joins), so they provide significant opportunities for our approach. Because our query engine does not support aggregate functions, GROUP BY and ORDER BY clauses, or subqueries, we have slightly modified the original queries. It should be noted that our goal in using TPC-H queries is to allow us to examine our approach using realistic join graphs, cardinalities, and selectivities.

Two transfer rates were arbitrary chosen for experiments: 1Mbytes/s and 2Mbytes/s, and two memory allocations for join operations: 2MB (small allocation - SA), 8MB (medium allocation - MA). The algorithm was implemented in Java 1.4 and experiments were conducted on PC Intel 1000Mhz, 512MB RAM under control of MS Windows 2000.

4.1. Experiment 1 – National Market Share

We start by studying the performance of our approach when delays are encountered during the execution of a modified version of TPC-H Query Q8, the National Market Share Query (referred to as MQ8). The SQL statement for MQ8 is shown in Figure 5.

MQ8 is an 8-way join query, with selections on the REGION, ORDER, and PART sources. We delay PART, which is a very important source in this query. The PART source, because of its selection predicate, plays the role of a reducer for LINEITEM, the largest source in the schema. Results of the experiments are presented in Figures 6 and 7.

Figure 6 presents the system response time for the query MQ8 versus initial delay of the PART for different values of the transfer rate and different

```
SELECT o.orderdate, l.extendedprice, n2.name
FROM part p, customer c, order o, lineitem l, supplier s,
     nation n1, nation n2, region r
WHERE  p.partkey = l.partkey
     and l.suppkey = s.suppkey
     and o.orderkey = l.orderkey
     and c.custkey = o.custkey
     and c.nationkey = n1.nationkey
     and n1.regionkey = r.regionkey
     and r.name = 'europe'
     and s.nationkey = n2.nationkey
     and o.orderdate between '94-01-01' and '95-12-31'
     and p.type = 'small plated steel'
```

Figure 5. MQ8: National Market Share

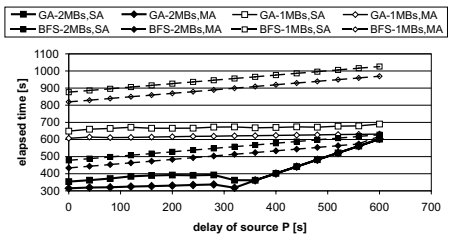


Figure 6. Elapsed time of MQ8 execution versus initial delay of PART

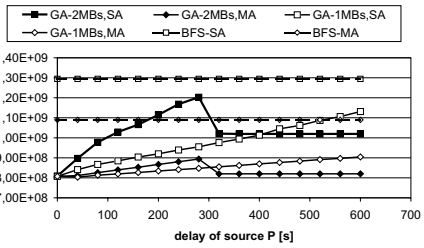


Figure 7. Processed data volume by MQ8 versus initial delay of PART

memory allocations. For 2Mbytes/s transfer rate, we observe that the algorithm switches from QEP1 (Figure 8) to QEP2 (Figure 9) for 320 seconds delay. This switch appears when the delay in access to PART is long enough for the execution of the operator 6 of QEP2 to finish before the algorithm collects statistically reliable samples from the execution of operator 5 of QEP1. After the switch we may observe a short-lived decrease in the response time of competition-based strategy. This phenomenon can be explained as follows: simultaneous execution of the whole QEP1 and part of QEP2 takes a bit more time than the execution of only QEP2. Next, we observe an increase in the response time which is proportional to PART's delay - operator 6 of QEP2 waits for source PART. For 1Mbytes/s transfer rate, we do not observe any switch. QEP1 always wins the competition. Small memory allocation increases response time, because operator 5 of QEP1 and operator 5 of QEP2 perform external partitioning to process the largest source LINEITEM. From the figure follows that the competition-based strategy outperforms the "brute-force" strategy.

Figure 7 presents the volume of processed data (i.e. overhead) versus the initial delay of PART and different values of the transfer rate. The overhead depends on the delay in access to most attractive sources. Increasing the elay

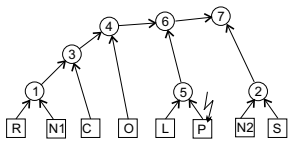


Figure 8. QEP1 of MQ8

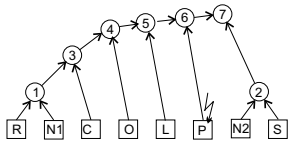


Figure 9. QEP2 of MQ8

```
SELECT n.name, o.orderdate.year,
     l.extendedprice*(1-l.discount) -
     (ps.supplycost * l.quantity)
FROM part p, supplier s, lineitem l, partsupp ps,
     order o, nation n
WHERE s.suppkey = l.suppkey
     and ps.suppkey = l.suppkey
     and ps.partkey = l.partkey
     and p.partkey = l.partkey
     and o.orderkey = l.orderkey
     and s.nationkey = n.nationkey
     and p.name like '%magenta%'
```

Figure 10. MQ9: Product Type Profit Measure

in data transfer from the PART delays the moment of competition termination. Thus, it extends the execution time of QEPs which belong to a competition group and increases the consumption of resources (disk I/O, CPU) consumption. The largest overhead is observed for 2Mbytes/s transfer rate, while the smallest is observed for 1Mbytes/s transfer rate. This phenomenon can be explained as follows: for a given delay of the source PART, in case of higher transfer rate, a large part of potentially unattractive subplans (e.g. QEP2) will be executed until the competition process stops their processing. In case of a lower transfer rate, unattractive subplans consume less resources since the algorithm cancels their processing "earlier". For 2Mbytes/s transfer rate, the overhead decreases after the switch from QEP1 to QEP2 because simultaneous execution of the whole QEP1 and a part of QEP2 takes more resources than the execution of single QEP2. We can eliminate or strongly reduce performance impact of this phenomenon by slight modification of the competition criterion. The competition criterion can be defined as the data volume that are necessary to terminate given competitor. Notice that competition-based strategy is cheaper than the "brute-force" strategy.

4.2. Experiment 2 – Product Type Profit Measure

We now turn to our second set of experiments, which uses a modified version of TPC-H Query 9, the Product Type Profit Measure (referred to as MQ9). This query is a 6-way join with one selection predicate. The query graph of MQ9 contains a cycle, unlike the "chain" graph of MQ8 in the previous experiment. The SQL for this query is shown in Figure 10.

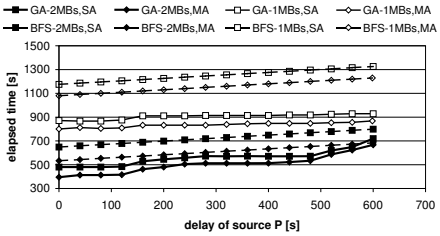


Figure 11. Elapsed time of MQ9 execution versus initial delay of PART

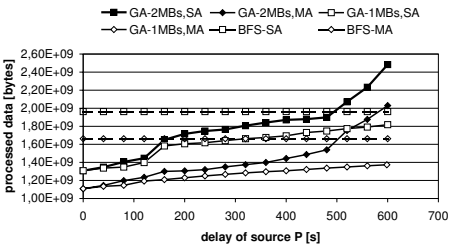


Figure 12. Processed data volume by MQ9 versus initial delay of PART

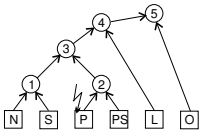


Figure 13. QEP3 of MQ9

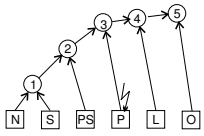


Figure 14. QEP4 of MQ9

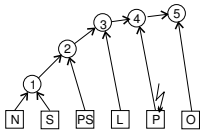


Figure 15. QEP5 of MQ9

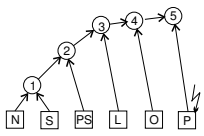


Figure 16. QEP6 of MQ9

We delay PART, which plays the role of a reducer for LINEITEM. Results of the experiments are show in Figures 11 and 12. Figure 11 presents the system response time for the query MQ9 versus the initial delay of the source PART for different values of the transfer rate and memory allocation. For 2Mbytes/s transfer rate, we may observe that the competition-based strategy switches from QEP3 (Figure 13) to a more expensive plan QEP4 (Figure 14) for 160 seconds delay. Next, we observe a switch from QEP4 to the most expensive QEP5 (Figure 15) for 520 seconds delay. After the last switch we observe the increase in the response time, which is proportional to the PART's delay because operator 4 of QEP5 waits for the source PART. For 1Mbytes/s, the GA algorithm switches between QEP4 and QEP5 for 160 seconds delay of source PART. Due to a smaller transfer rate we do not observe the second switch. Small memory allocation increases the response time, because operator 4 of QEP3, operator 4 of QEP4, and operator 3 of QEP5 perform external partitioning to process the largest source LINEITEM. From the figure follows that the competition-based strategy outperforms "brute-force" strategy.

Figure 12 presents the dependence of the volume of processed data of Q1 execution on the initial delay of the source PART. For 2Mbytes/s transfer rate, after the switch from QEP4 to QEP5 the overhead increases proportionally to the delay of PART. This increasing overhead is caused by the competition between operator 4 of QEP5 and operator 4 of a very expensive QEP6.

For 2Mbytes/s transfer rates, before the algorithm switches from QEP4 to QEP5, the competition strategy consumes less CPU time than the "brute-force" strategy. After the switch, the competition-based strategy processes a bit more data than the "brute-force" strategy. For 1Mbytes/s transfer rate, the competition-based strategy always processes less data than the "brute force"

strategy.

We also performed a series of experiments which tested different transfer rates for other data sources. We observed the correlation between the transfer rate of attractive data sources and the competition overhead. If attractive data sources transfer data with higher rate than other sources, then the competition overhead decreases. We observed also, that the competition strategy prefers bushy plans, which usually have shorter response times than linear plans. Due to the lack of space, we omit the detailed description of these experiments.

5. Summary and future work

In this paper we have proposed a novel strategy of query executing in the Web environment. Our strategy copes with limited availability of data statistics and unexpected delays in access to data sources. The strategy is based on the competition of simultaneously processed query execution plans. Dynamically collected statistics allow canceling the most expensive plans and promote the most promising plans. In this paper we also present the greedy algorithm which implements our strategy. We have evaluated our strategy by a set of experiments for different transfer rates, different main memory allocations, and different delay scenario, and we have proved its feasibility. As the experiments show, our strategy is especially appropriate for small and medium transfer rates. The strategy is efficient also for large transfer rate (2Mbytes/s) and relatively small delays in access to attractive sources. The algorithm can generate bushy QEPs, which, when compared to linear QEPs produced by traditional cost-based optimization algorithms, usually provide better response times.

The proposed algorithm has two shortcomings. First, due to nature of the algorithm, it generates suboptimal plans. Second, the algorithm materializes partial results of subplans which may increase the cost of the optimization process. These shortcomings can be eliminated, or at least their impact can be restricted, if we assume that the algorithm initiates only complete query execution plans. In order to restrict the number of simultaneously processed plans, we can perform preliminary optimization, which could use partial statistics collected during previous executions of similar queries. Our future work includes enhancements of the presented algorithm to improve its performance. Another topic, which we would like to address in the near future, is the issue of generation of query execution plans called parachute QEPs constructed on the basis of partial results produced by the initial set of plans.

References

[1] Gennady Antoshenkov. Dynamic query optimization in rdb/vms. In *Proceedings of the Ninth International Conference on Data Engineering, April 19-23, 1993, Vienna, Austria*, pages 538–547. IEEE Computer Society, 1993.

[2] Ron Avnur and Joseph M. Hellerstein. Eddies: Continuously adaptive query processing. In *Proceedings of the 2000 ACM SIGMOD International Conference*

on Management of Data, May 16-18, 2000, Dallas, Texas, USA, volume 29, pages 261–272. ACM, 2000.

[3] Anastasis Gounaris, Norman W. Paton, Alvaro A.A Fernandes, and Rizos Sakellariou. Adaptive query processing: A survey. In *Advances in Databases, 19th British National Conference on Databases, BNCOD 19, Sheffield, UK, July 17-19, 2002, Proceedings*, volume 2405 of *Lecture Notes in Computer Science*. Springer, 2002.

[4] Peter J. Haas and Joseph M. Hellerstein. Ripple joins for online aggregation. In *SIGMOD 1999, Proceedings ACM SIGMOD International Conference on Management of Data, June 1-3, 1999, Philadelphia, Pennsylvania, USA*, pages 287–298. ACM Press, 1999.

[5] Peter J. Haas, Jeffrey F. Naughton, S. Seshadri, and Arun N. Swami. Fixed-precision estimation of join selectivity. In *Proceedings of the Twelfth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, May 25-28, 1993, Washington, DC*, pages 190–201. ACM Press, 1993.

[6] Zachary G. Ives, Alon Y. Levy, Daniel S. Weld, Daniela Florescu, and Marc Friedman. Adaptive query processing for internet applications. *Bulletin of the Technical Committee on Data Engireening - Special Issue on Adaptive Query Processing*, 23(2):19–26, 2000.

[7] Navin Kabra and David J. DeWitt. Efficient mid-query re-optimization of sub-optimal query execution plans. In *SIGMOD 1998, Proceedings ACM SIGMOD International Conference on Management of Data, June 2-4, 1998, Seattle, Washington, USA*, pages 106–117. ACM Press, 1998.

[8] Gang Luo, Curt J. Ellmann, Peter J. Haas, and Jeffrey F. Naughton. A scalable hash ripple join algorithm. In *Proceedings of the ACM SIGMOD international conference on Management of data*, pages 252–262. ACM Press, 2002.

[9] Fatma Ozcan, Sena Nural, Pinar Koksal, Cem Evrendilek, and Asuman Dogac. Dynamic query optimization in multidatabases. *Data Engineering Bulletin*, 20(3):38–45, 1997.

[10] Tolga Urhan and Michael J. Franklin. Xjoin: A reactively-scheduled pipelined join operator. *Bulletin of the Technical Committee on Data Engireening - Special Issue on Adaptive Query Processing*, 23(2):27–33, 2000.

[11] Tolga Urhan, Michael J. Franklin, and Laurent Amsaleg. Cost based query scrambling for initial delays. In *SIGMOD 1998, Proceedings ACM SIGMOD International Conference on Management of Data, June 2-4, 1998, Seattle, Washington, USA*, pages 130–141. ACM Press, 1998.

[12] Stratis Viglas, Jeffrey F. Naughton, and Josef Burger. Maximizing the output rate of multi-way join queries over streaming information sources. In *VLDB 2003, Proceedings of 29th International Conference on Very Large Data Bases, September 9-12, 2003, Berlin, Germany*, pages 285–296. Morgan Kaufmann, 2003.

[13] Annita N. Wilschut and Peter M. G. Apers. Dataflow query execution in a parallel main-memory environment. In *Proceedings of the First International Conference on Parallel and Distributed Information Systems (PDIS 1991), Fontainebleu Hilton Resort, Miami Beach, Florida, December 4-6, 1991*, pages 68–77. IEEE Computer Society, 1991.

SCIENTIFIC PAPERS UNIVERSITY OF LATVIA. 2004. VOLUME 672, Datbases and Information Systems
93–108 P.

Enhancing Hierarchical Queries in Relational Databases with the Nested Set Representation

Lauri Pietarinen

Relational Consulting Oy
Vuorimiehenkatu 15 A 3,
00140 Helsinki
Finland
lauri.pietarinen@relational-consulting.com

Boris Novikov

University of St.-Petersburg
Universitetsky prosp. 28
198904 St.-Petersburg
Russia
borisnov@acm.org

Abstract. Queries on hierarchical structures have traditionally been the sore point of relational databases because of missing functionality in SQL. Simple tree traversal is not possible without procedural code or vendor specific extensions. Using the nested set representation to represent trees vendor neutral SQL can be used to express ancestor and descendant queries. In our paper we propose enhancements to the original nested set representation to speed up tree traversal queries and enable extra functionality, such as 'by level' queries. We also introduce an algorithm for inserting new nodes into the tree without having to renumber existing ones. We also show how ancestor queries can be optimized using our enhancements. Finally we compare the performance of this new representation with that of the original representation and to the recursive union as implemented in DB2.

Keywords. Hierarchical query, SQL, Nested Set, tree, subtree

1. Introduction

Hierarchical structures are ubiquitous in information systems. We can distinguish between hierarchies in schema, such as a schema involving the entities department, group, team and person, and hierarchies in which the nodes are of the same kind. We can further distinguish between hierarchies with potential cycles, such as bill-of-material type hierarchies in which children can have more than one parent and strict hierarchies in which each node has exactly one parent except for the root node which has no parents [11].

Examples of strict hierarchies include organizational hierarchies, product group hierarchies, newsgroup and email threads, web site maps, file folder structures, project task hierarchies, word hierarchies in thesauri [4] and XML-hierarchies [7],[9].

The traditional and obvious way to represent hierarchies (and arbitrary graphs) in relational databases is the adjacency list approach in which the child node refers to its parent node with its primary key. In spite of this, SQL has not traditionally provided good support for querying hierarchical (or other recursive) structures apart from some vendor specific extensions such as Oracle's Connect By.

The nested set representation [1], [2], [6] to represent hierarchies uses a partly materialized non-recursive structure to improve query performance with a degrading of update performance. While it is not as general as the adjacency list representation, which can be used for arbitrary graphs, it is a good alternative for situations where strict, possibly ordered trees are used. It is based on a numbering scheme in which the nodes are tagged with their postorder and preorder (or visitation order) in the hierarchy.

Because of the non-recursiveness of the nested set representation, queries such as

- get nodes of subtree in preorder
- get ancestors of node
- get closure of tree

are easy to express and efficient to execute using standard SQL available on any SQL-based database.

In this paper we propose modifications to the nested set representation for enhanced performance and functionality and compare this enhanced representation with the recursive union queries available in DB2.

We also introduce new queries to overcome deficiencies in the original nested set approach, especially in ancestor searches over large hierarchies.

The paper proceeds as follows. The next section provides an overview of related work. In Section 3 we describe the original nested set model giving examples in SQL. In section 4 we describe our enhancements for the model. In Section 5 we present some typical use cases with an eye on indexes for optimizing performance. In Section 6 we compare the functionality and performance of our approach with that of DB2 Recursive Union and then conclude with some further suggestions.

2. Related Work

The best known technique to handle hierarchies is based on the edge representation of graphs, where an edge of a graph is stored as a pair of starting and ending nodes. Extensive research in this area has been done, and several efficient query evaluation techniques were proposed in the context of deductive databases [11].

From the practical perspective, the disadvantages of this structure are the following:

- it is difficult to represent the order of nodes (say, child nodes), and
- processing of the hierarchies of indefinite depth requires recursive queries.

Although recursive queries are now available in SQL:1999, this feature is not yet widely supported by DBMS vendors. Additionally, since the result of the recursive union query does not contain order information it is not possible to further join the result without losing the tree structure. Further, the recursive union construct does not enable the output of ordered hierarchies, such as XML documents or newsgroup threads.

The nested set representation is described in [1] in general terms without any references to databases. The application to relational databases can be found in [2] and [6].

Although the nested set representation has turned out to be useful in practice [4] not much attention has been given to it in the scientific community and hence our primary references are to books or vendor user group presentations.

Similar representations are discussed in [7], [8] and [12] but they are either more complex or rely on vendor specific optimization techniques, such as index-anding, or propose modifications to the database optimizer and engine.

In [13] a variation of the nested set representation is described that uses a binary rational numbering.

3. The Nested Set Representation

We now describe the background ideas of the nested set representation and continue with an SQL-based example. We then present some SQL-queries that make use of that example.

3.1 Fundamentals

One way to represent hierarchies is to use nested sets [1]. Each node of the hierarchy is represented as a set. Relationships between nodes are modeled using set inclusions. Sets corresponding to leaf nodes should be disjoint. The elements of these sets are not important for us. For simplicity we assume that sets corresponding to leaf nodes contain single elements, which are all distinct.

The set containing all the other sets is the root of the tree and the sets containing no subsets are the leaf nodes of the tree. The ancestors of a node are the nodes that contain this node and the descendants of a node are the nodes that this node contains. We require that for any two different sets, either one is the proper subset of the other or their intersection is empty. Such a hierarchy is clearly unordered, i.e. the children under a given parent are not in any specific order.

The example hierarchy in figure 1 could be represented with nested sets as in figure 2.

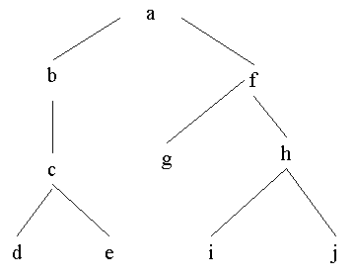


Figure 1 Sample hierarchy

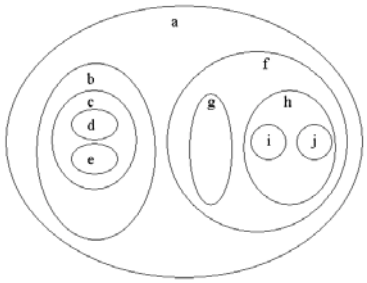


Figure 2 Sample hierarchy using nested sets

The applications we are considering, such as XML and newsgroups the immediate descendants (child nodes) of any node are usually ordered.

The ordering of child nodes is achieved by changing the representation to using sets of closed intervals in an (ordered) space of rational numbers. To obtain these intervals, we assign a rational number to each bracket (see below) so that the ordering of the numbers corresponds to the order of the brackets regardless of whether it is a left or right bracket. Clearly the properties previously stated still apply. Additionally we can see that the children of each node are ordered.

Using figure 1 as an example we obtain

```
A [ B [ C [ D [ ] E [ ] ] ]
  Q1  Q2   Q3   Q4  Q5   Q6 Q7 Q8 Q9
F [ G [ ] H [ I [ ] J [ ]  ] ]  ]
 Q10   Q11Q12   Q13 Q14Q15   Q16Q17Q18Q19 Q20

WITH QI ⊂ Q AND Q1 < Q2 < … < Q20.
```

In this example we can see that the children of `h` are in the order `(i,j)` .

In this paper we adopt the convention of calling the lower bound (the left bracket) of a set the `left bound` and the upper bound (the right bracket) of the set the `right bound`. For example, the left bound and right bound of node f are 10 and 19 respectively. It can easily be seen that the left and right bounds correspond to the preorder and post order of the nodes in the hierarchy [3].

By using consecutive integer numbers instead of ordered rational numbers we further obtain the property that the number of descendants of any given node can be calculated with the formula (`RIGHT BOUND − LEFT BOUND`-1)/2. As a corollary the leaf nodes have (`RIGHT BOUND−LEFT BOUND`-1)/2 = 0.

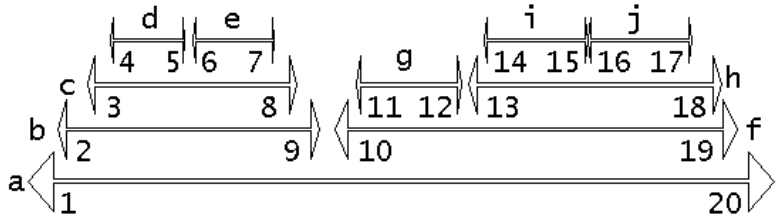
We adopt the convention of assigning the left bound of the root of the hierarchy to 1. The right bound of the root will be equal to the total number of nodes in the hierarchy times two.

Returning to our example we would have the following boundaries:

```
A [ B [ C [ D [ ] E [ ] ] ]
  1   2   3   4  5     6  7  8  9
F [ G [ ] H [ I [ ] J [ ]  ] ]  ]
  10    11 12    13  14 15  16 17  18  19 20
```

We can see that the number of children of node F is (19-10-1)/2 = 4.

Figure 3 describes an alternative way to visualize the example hierarchy.



:

Figure 3 Sample hierarchy depicted as nested line segments

3.2 Representation in SQL

We now continue with an example in SQL. To store hierarchy information we create the following table:

```
CREATE TABLE TREETAB
 (NODE_ID CHAR(1) PRIMARY KEY,
  LEFT_BOUND INT NOT NULL,
  RIGHT_BOUND INT NOT NULL)
```

The sample hierarchy of figure 1 is represented by the values in table 1.

Node id	**Left bound**	**Right Bound**
a	1	20
b	2	9
c	3	8
d	4	5
e	6	7
f	10	19
g	11	12
h	13	18
I	14	15
j	16	17

Table 1

To ensure that in all cases the value of the table is consistent with our abstract representation we need the following integrity constraints:

1. For all nodes, left bound < right bound
2. No overlapping intervals
3. Existence and uniqueness of root node
4. Left bound of root equal to 1
5. Consecutive numbering

3.3 Sample queries

Using the properties of the representation as stated in section 3.1 we can now formulate some sample queries.

The descendants of the node f can be listed in preorder with the following query:

```
SELECT NODE_ID,
       LEFT_BOUND
  FROM TREETAB
  WHERE LEFT_BOUND > 10 AND
        LEFT_BOUND < 19
  ORDER BY LEFT_BOUND;
```

The ancestors of 'j' can be listed from root downwards with the following query:

```
SELECT NODE_ID,
       LEFT_BOUND
  FROM TREETAB
  WHERE LEFT_BOUND  < 16 AND
        RIGHT_BOUND > 17
  ORDER BY LEFT_BOUND;
```

4. Enhancing the Nested Set Representation

The original representation clearly suffers from problems when new nodes are added to a large hierarchy. Because of the dense numbering adding a new node to the hierarchy results in updating on the average half of all rows. Our first enhancement is to introduce sparse numbering.

The second enhancement is related to the level of the node in the hierarchy.

We proceed by proposing the following improvements to the nested set representation:

1. using sparse numbering to speed up inserts
2. adding the level of each node to the representation

4.1 Speeding up Inserts

One problem with the nested set representation as described in [2] is that when inserting a new node into the tree on average half the nodes need to be updated. This is because the node numbering is dense. For example inserting a new child under 'b' would result in updating the numbering of the nodes 'f', 'g', 'h', 'i', 'j' and 'a'.

With small trees this is not a great problem, but as the amount of nodes increases the amount of updates increases linearly.

The obvious solution to this is to introduce a sparse numbering scheme so that most of the times the insertion of a new node can be done without updating other nodes. The property that is lost is the simple calculation to obtain the number of nodes in a subtree (including the "is leaf" -property).

Naturally there comes a time when a new node does not fit into its designated place and the nodes of the tree have to be renumbered. This renumbering can be

done during idle time or on demand. Further, the renumbering can be done locally for "quick relief" or globally for a more lasting effect.

We introduce the simple algorithms ADD_NODE and RENUMBER_ALL. ADD_NODE is optimized for newsgroup type trees where new nodes are always inserted as next (or first) child of parent. RENUMBER_ALL renumbers the bounds so as to use the available space as sparsely as possible.

```
ALGORITHM RENUMBER_ALL

  CONST MAX_INTEGER = 2**31-1

  NODE_COUNT :=
     SELECT COUNT(*) FROM TREE;

  RUNNING_BOUND  := 0;
  BOUND_INTERVAL :=
   MAX_INTEGER/(NODE_COUNT*2)-1;

  FOR EACH
      SELECT NODE_ID,
             LEFT_BOUND AS BOUND_VALUE,
             'LEFT' AS BOUND_TYPE
      FROM TREE
      UNION ALL
      SELECT NODE_ID,
             RIGHT_BOUND AS BOUND_VALUE,
             'RIGHT' AS BOUND_TYPE
      FROM TREE
      ORDER BY BOUND_VALUE

    IF BOUND_TYPE = 'LEFT' BEGIN
      RUNNING_BOUND = RUNNING_BOUND + 1
      UPDATE TREE
        SET LEFT_BOUND = :RUNNING_BOUND
        WHERE NODE_ID = :NODE_ID;
      END
    ELSE BEGIN
      RUNNING_BOUND :=
          RUNNING_BOUND + BOUND_INTERVAL
      UPDATE TREE
        SET RIGHT_BOUND = :RUNNING_BOUND
        WHERE NODE_ID = :NODE_ID;
      END
  END
```

```
ALGORITHM ADD_NODE ( PARENT_NODE_ID,
                     NODE_ID )

    -- GET PARENT NODE INFO

    SELECT LEFT_BOUND,
           RIGHT_BOUND,
           NODE_LEVEL
    FROM TREE
    WHERE NODE_ID =
         :PARENT_NODE_ID;

    -- GET RIGHT-MOST CHILD INFO
     SELECT MAX(RIGHT_BOUND) AS
            MAX_RIGHT_BOUND
     FROM TREE
     WHERE RIGHT_BOUND <
          :RIGHT_BOUND AND
           RIGHT_BOUND >
            :LEFT_BOUND
     IF NOT_FOUND THEN
       MAX_RIGHT_BOUND := LEFT_BOUND

     -- IS THERE ROOM? IF NOT,
      -- RENUMBER AND REFETCH
     -- BOUND-INFORMATION

     IF RIGHT_BOUND-LEFT_BOUND <=2
       BEGIN
         EXECUTE RENUMBER_ALL
         SELECT MAX(RIGHT_BOUND) AS
                MAX_RIGHT_BOUND
           FROM TREE
           WHERE RIGHT_BOUND <
                 :RIGHT_BOUND AND
                 RIGHT_BOUND >
                 :LEFT_BOUND
         IF NOT_FOUND THEN
            MAX_RIGHT_BOUND :=
                      LEFT_BOUND
       END

     NEW_LEFT_BOUND  :=
         MAX_RIGHT_BOUND + 1;
     NEW_RIGHT_BOUND :=
         MAX_RIGHT_BOUND +
          FLOOR( (1/3) *
       (RIGHT_BOUND-NEW_LEFT_BOUND) );
     NEW_LEVEL := NODE_LEVEL + 1;

     INSERT INTO TREE
          (NODE_ID, LEFT_BOUND,
           RIGHT_BOUND, NODE_LEVEL)
     VALUES
         (:NODE_ID, :NEW_LEFT_BOUND,
         :NEW_RIGHT_BOUND, :NEW_LEVEL)
```

4.2 Adding the Hierarchy Level to Each Node

The level of each node can be derived from the number of ancestors it has. We adopt the convention that the level of the node is the number of proper ancestors + 1, i.e. the root has level 1, the children of the root have level 2 etc…

The level of each node can be calculated by joining the tree table with itself and counting the ancestors:

```
SELECT TREE1.NODE_ID,
       COUNT(*) AS NODE_LEVEL
  FROM TREETAB TREE1,
       TREETAB TREE2
  WHERE TREE1.LEFT_BOUND  >= TREE2.LEFT_BOUND AND
        TREE1.RIGHT_BOUND <= TREE2.RIGHT_BOUND
  GROUP BY TREE1.NODE_ID
```

This is clearly an inefficient and cumbersome way to produce the level information.

We propose that a new attribute, 'level', be added to each node. The level can be easily calculated when the new node is inserted (see algorithm ADD_NODE). We have found no performance penalties for doing this, apart from a small increase in storage space.

Further on we shall see in section 6 that this new attribute is of use when we have to query efficiently for ancestors of a given node in a large tree.

The create statement for the new table is

```
CREATE TABLE TREETABL
 (NODE_ID CHAR(1) PRIMARY KEY,
  LEFT_BOUND INT NOT NULL,
  RIGHT_BOUND INT NOT NULL,
  NODE_LEVEL INT NOT NULL)
```

5. Typical Use Cases and their Access Paths

We will now present some typical queries and describe the access paths using normal B-tree indexes available in any SQL-database. Our aim is to display the best access paths (i.e. the access paths that result in the fastest execution) that are obtainable in each case.

Table 2 containes the B-tree indexes that are used in the queries.

Name	Unique	Columns
Xtree_node	X	node_id
Xtree_left	X	left_bound
Xtree_leftd	X	left_bound desc, right_bound
Xtree_right	X	right_bound
Xtree_lev_left	X	node_level, left_bound
Xtree_lev_leftd	X	node_level, left_bound desc

Table 2 Indexes for typical use cases

5.1 Subtree in Pre- and Post-Order

We obtain a subtree in preorder with the following query:

```
SELECT TREE2.NODE_ID,
       TREE2.NODE_LEVEL,
       TREE2.LEFT_BOUND
  FROM TREETABL TREE1,
       TREETABL TREE2
  WHERE TREE1.NODE_ID = <SUBTREE ROOT> AND
        TREE2.LEFT_BOUND >= TREE1.LEFT_BOUND AND
        TREE2.LEFT_BOUND <= TREE1.RIGHT_BOUND
  ORDER BY LEFT_BOUND
```

Using the indexes XTREE_NODE and XTREE_LEFT we are clearly obtaining an optimal access path: first a lookup on the root id of the sub tree and then a range scan on index xtree_left with no need for sorting. The access path can be portrayed as in figure 3.

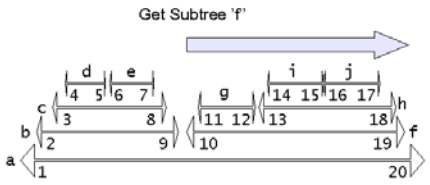


Figure 3 Get subtree of node F

The subtree in postorder is obtainable simply by replacing LEFT_BOUND with RIGHT_BOUND in the order by clause. The optimizer can then choose the index XTREE_RIGHT instead.

5.2 Get Children of a Given Node

```
SELECT TREE2.NODE_ID,
       TREE2.NODE_LEVEL,
       TREE2.LEFT_BOUND
  FROM TREETABL TREE1,
       TREETABL TREE2
  WHERE TREE1.NODE_ID = <PARENT_ID> AND
        TREE2.LEFT_BOUND > TREE1.LEFT_BOUND AND
        TREE2.LEFT_BOUND < TREE1.RIGHT_BOUND AND
        TREE1.NODE_LEVEL+1 = TREE2.NODE_LEVEL
  ORDER BY LEFT_BOUND
```

The query is essentially the same as in 5.1 except for the restriction on NODE_LEVEL. Using indexes XTREE_NODE and XTREE_LEV_LEFT an optimal access path is obtained.

5.3 Ancestors of a Given Node

While the actual query is easy to state, obtaining an acceptable access path is not so trivial. We will start with the obvious solution:

```
SELECT TREE2.NODE_ID,
       TREE2.NODE_LEVEL,
       TREE2.LEFT_BOUND
  FROM TREETABL TREE1,
       TREETABL TREE2
  WHERE TREE1.NODE_ID = <NODE_ID> AND
        TREE2.LEFT_BOUND  < TREE1.LEFT_BOUND AND
        TREE2.RIGHT_BOUND > TREE1.RIGHT_BOUND
  ORDER BY LEFT_BOUND
```

Using the indexes XTREE_NODE and XTREE_LEFTD we can see that the amount of nodes that have to be examined depends on the position of the node. On the average, half of all nodes of the whole tree must be traversed. Hence the time to obtain the ancestors of a given node rises linearly with the number of nodes in the tree. As can be seen in figure 4, finding the ancestors of 'j' requires searching through nearly the whole tree.

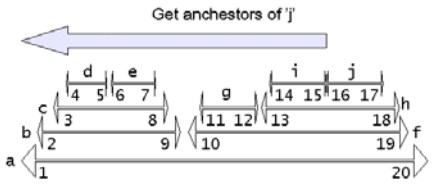


Figure 4

We will return to this performance problem and present an elegant solution in section 6.

5.4 The Transitive Closure

Due to limitation of the data structure to represent only strict hierarchical trees (rather than arbitrary graphs), the transitive closure is effectively equivalent to the selection of a subtree. However, we would like to produce the output in ordinary relational form, compatible with the output from recursive queries.

The closure of the tree can be obtained by the following query:

```
SELECT TREE1.NODE_ID,
       TREE1.NODE_LEVEL,
       TREE1.LEFT_BOUND,
       TREE2.NODE_LEVEL,
       TREE2.NODE_LEVEL -  TREE1.NODE_LEVEL AS DISTANCE,
       TREE2.LEFT_BOUND
  FROM TREETABL TREE1,
       TREETABL TREE2
  WHERE TREE1.LEFT_BOUND <= TREE2.LEFT_BOUND AND
        TREE1.RIGHT_BOUND > TREE2.LEFT_BOUND
```

Remarks for this query are essentially the same as for the 'Get subtree'- query stated previously.

6. The 'get ancestors' problem revisited

The solution to the performance problem for 'get ancestors' query relies on an observation that a node has at most one ancestor on each level. For each of the levels it is possible to obtain that ancestor efficiently. It is the node with the largest left_bound that is smaller than the left_bound of the given node and the level less than the level of the given node.

E.g. node 'j' has one ancestor on each of the levels 1, 2 and 3 (a, f and g respectively). Hence f has the largest left_bound, that is smaller than the left_bound of 'j', than any other node with the same level (which is 2).

This is a query that is efficiently expressible in SQL using standard B-trees and a modern cost based optimizer:

```
SELECT TREE2.NODE_ID,
       TREE2.NODE_LEVEL,
       TREE2.LEFT_BOUND
  FROM TREETABL TREE1,
       TREETABL TREE2
  WHERE TREE1.NODE_ID = <NODE_ID> AND
        TREE2.LEFT_BOUND =
         ( SELECT MAX(LEFT_BOUND)
             FROM TREETABL TREE3
             WHERE TREE3.NODE_LEVEL = 2 AND
             TREE3.LEFT_BOUND <=  TREE1.LEFT_BOUND )
```

Using the indexes XTREE_NODE and XTREE_LEV_LEFTD (see table 2) the access path will be nested matching index scans on both indexes.

To expand the query to obtain all ancestors we introduce an auxiliary table [5] for enumerating the levels:

```
CREATE TABLE TABLENUM
 ( N INTEGER PRIMARY KEY );
```

and insert the values 1..x into it (x being the maximum level we presume the tree to have).

We now modify the query slightly to obtain each of the ancestors separately for each level that is less than the level of the given node:

```
SELECT TREE2.NODE_ID,
       TREE2.NODE_LEVEL
  FROM TREETABL TREE1,
       TREETABL TREE2,
       TABLENUM NUM
  WHERE TREE1.NODE_ID = <NODE_ID> AND
        NUM.N < TREE1.NODE_LEVEL AND
        TREE2.LEFT_BOUND =
         (SELECT MAX(LEFT_BOUND)
           FROM TREETABL TREE3
           WHERE TREE3.NODE_LEVEL =  NUM.N AND
                 TREE3.LEFT_BOUND <= TREE1.LEFT_BOUND )
```

In spite of the complex nature of the query the optimizers of both DB2 V8.1 and SQLServer 2000 will handle the query as expected. We will examine the performance of this query with the previous version in section 7.2.

7. Comparison and Evaluation

The comparison is made using the enhanced nested set representation, proposed in this work, with the "traditional" adjacency list representation and the original nested set representation with dense numbering. For performance evaluation purposes synthetic data sets were generated. The iterative generator adds the child nodes to random parents. In the sparse numbering scheme we used a space of 1 to 2^{31}-1. The generated trees and their properties can be seen in table 1.

Tree	Number of nodes	Height of tree
1	1000	17
2	2000	18
3	5000	20
4	10000	22
5	20000	22
6	50000	25
7	100000	26
8	200000	27

Table 3 Generated hierarchies

Performance tests were done using IBM DB2 UDB V8.1 on a computer with a 500MHz Pentium P3 processor and Microsoft Windows 2000 Professional operating system. All measurements are in milliseconds.

7.1 Building the Tree: the Performance of Inserts

The first performance tests concern the insert speed of nodes using either dense or sparse numbering. As can be expected, the sparse numbering scheme outperforms the dense alternative on inserts. Actually, the insertion time grows linearly with the tree size for dense numbering and remains almost constant for sparse numbering.

Our tests include 10 runs for both sparse and dense numbering. In chart 1 we can see how renumbering requests are done periodically for the sparse tree, on average three times per test run. Other than for the renumbering, the insert speed for the nodes remains constant using the sparse numbering, at about 3,5 ms per node added.

Naturally in some situations, such as loading XML-documents into a database [7], the bounds of the nodes can be precalculated in which case dense numbering will perform well. It is mainly in random inserts that the sparse numbering will bring benefits.

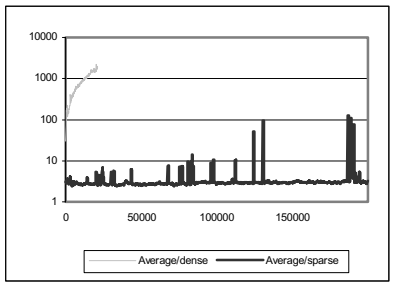


Chart 1. Inserting nodes(ms/size of tree)

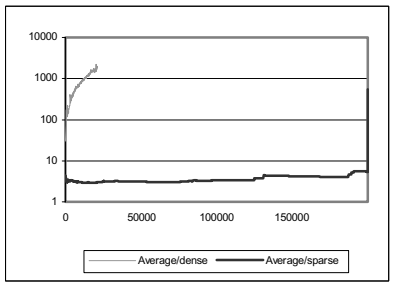


Chart 2. Inserting nodes, averaged (ms/size of tree)

Chart 2 is averaged over the whole space and we can see the total average time slowly rising.

The problem, of course, with sparse numbering is that occasionally a renumbering is required and if it is done for the whole tree it creates a random delay that may not be acceptable. In chart 3 we see how the node count affects the time needed to renumber the hierarchy.

A practical solution to this problem would be to introduce local renumbering algorithms resulting in less frequent global reorganizations. However, this enhancement has not been done in this research.

Another solution is to do the renumbering periodically during idle time. A larger numbering space (e.g. 64 bit integer) will also lessen the need for renumbering.

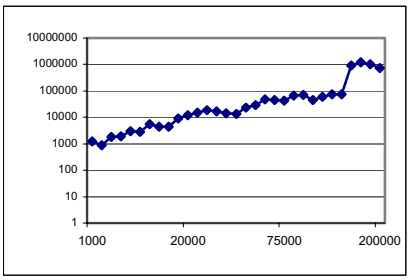


Chart 3: Time needed to renumber hierarchy (ms/size of tree)

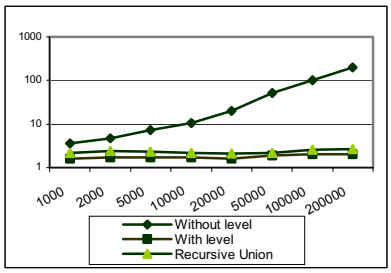


Chart 4. Get ancestors (ms/size of tree)

7.2 Query Performance

To test the usefulness of adding the 'node_level' attribute we compared the performance of three queries:

1. Get ancestors
2. Get siblings
3. Get subtree nodes

Getting ancestors using the original query (see section 5.3) has to scan on average half of all nodes, hence performance degrades as the size of the hierarchy increases regardless of the depth of the tree. As can be seen in chart 4 the get ancestor-query using the SQL-statement introduced in section 6 clearly outperforms the non-optimized query and its performance does not depend on the size of the tree. What is surprising is that this enhanced query performs about the same or better than the recursive union version.

In the siblings query comparison in chart 5 we can observe that the query using node levels performs about an order of magnitude faster than the query without node levels materialized.

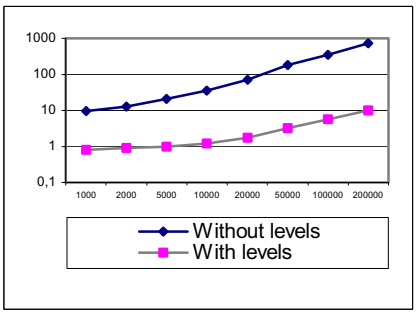


Chart 5. Get siblings (ms/size of tree)

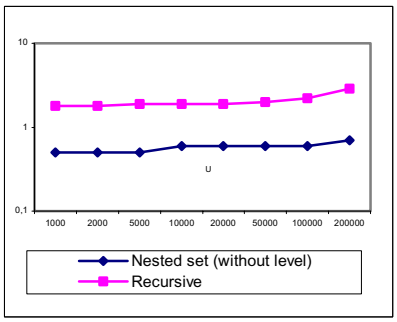


Chart 6. Get subtree in preorder (ms/size of tree)

The subtree query (see section 5.1) is faster than the recursive union version even without the proposed enhancements, as can be seen in chart 6.

8. Conclusions

We have shown that the nested set representation is useful in situations where strict ordered hierarchies have to be processed in preorder and/or postorder. By adopting the modifications proposed in this paper we can keep the advantages of the nested set representation and still remain competitive with the new SQL language constructs, such as recursive union.

The clearest advantage is in the fact that all queries against the nested set representation can be expressed in standard SQL available on any SQL-based database. The results of the expressions can be further joined to fact tables or to other hierarchies. Because of the numbering scheme we are not relying on any implicit output order for preordering or postordering such as with Oracle's Connect By extension, but can reorder and regroup the nodes at will without losing structure information.

The limits of the original representation as described in [2] and [6] can be overcome by adding sparse numbering and level information to the nodes. We can now get equivalent or better performance compared to the traditional adjacency list representation in queries such as 'get ancestors' and operations such as 'insert node'. It also adds to the types of queries that are possible to express.

Further work is needed to optimize the renumbering algorithm so that the renumbering time and cost can be contained within given limits.

References

[1] Donald E. Knuth. The Art of Computer Programming. Volume 1 pages 312, 317.

[2] Joe Celko. SQL for Smarties (2ed) Morgan Kaufmann Publishers, 1999.

[3] Aho, Alfred V., John E. Hopcroft, and Jeffrey D. Ullman. *The design and analysis of computer algorithms.* AddisonWesley: Reading, MA, 1974.

[4] Ballew, Duncan, Blasingame. Relational Data Structures for Implementing Thesauri http://www.mip.berkeley.edu/mip/related/thesaurus.html

[5] Hugh Darwen. A constant friend. In Relational Database Writings 1985-1989 by C. J. Date, pages 493-500. Prentice Hall, 1990.

[6] Leif Morten Kofoed and Håkon Erdal. Trees and SQL. Presentation at DB2 G.U.I.D.E . April 1995, Lahti Finland.

[7] Torsten Grust. Accelerating XPATH location steps. In. *Proc. ACM SIGMOD 2002.* P. 109-130.

[8] Chun Zhang, Jeffrey Naughton, David DeWitt, Qiong Luo, and Guy Lohman. On Supporting Containment Queries in Relational Database anagement Systems. In Proc. of

the ACM SIGMOD Int'l Conference on Management of Data, pages 425-436, Santa Barbara, California, May 2001. ACM Press.

[9] Quanzhong Li and Bongki Moon. Indexing and Querying XML Data for Regular Path Expressions. In Proc. of the 27th Int'l Conference on Very Large Data Bases (VLDB), pages 361-370, Rome, Italy, September 2001.

[10] Srini Venigalla. Expanding Recursive Opportunities with SQL UDFs in DB2 v7.2. http://www7b.boulder.ibm.com/dmdd/library/techarticle/0203venigalla/0203venigalla.html

[11] Ullman, Jeffrey. Principles of database and knowledge base systems. Vol. 2.. Academic Press, 1989.

[12] Torsten Grust, Maurice van Keulen, Jens Teubner. Staircase Join: Teach A Relational DBMS to watch it's (Axis) steps. In Proc. of the 29th Int'l Conference on Very Large Data Bases (VLDB), Berlin, Germany, September 2003.

[13] Vadim Tropashko: Trees in SQL: Nested Sets and Materialized path. http://www.dbazine.com/tropashko4.shtml

SCIENTIFIC PAPERS UNIVERSITY OF LATVIA. 2004. VOLUME 672, Datbases and Information Systems
109–121 P.

Automatic Generation of Minimal and Safe Transactions in Conceptual Database Design

M. A. Pastor, M. Celma-Giménez, L. Mota-Herranz

Departamento de Sistemas Informáticos y Computación;
Universidad Politécnica de Valencia
Camino de Vera s/n E-46022 Valencia – España
{mapastor, mcelma, lmota}@dsic.upv.es

Abstract. The conceptual design of information systems using an Entity-Relationship Model is generally thought to be limited to static features. Nevertheless, many dynamic aspects can be derived from an Entity-Relationship diagram since the specification of different constraints entails a determined minimal structure on the transactions that can be applied to such a system. In our work, we propose an algorithm that obtains, directly from the Entity-Relationship diagram, the set of minimal and safe transactions that can be performed on the system. An update (insert or delete) transaction on an object (entity type or relationship type) is said to be "safe" if it includes, besides the update operation, all other operations on any system object which are required to satisfy the integrity constraints. A safe transaction is said to be "minimal" if there is not a subset of the transaction which is safe too. The minimal and safe transactions are the basic units that can be used to specify the more complex transactions which model user requirements.

Keywords. Safe transaction generation, conceptual database design, integrity constraints.

1. Introduction

The *Entity-Relationship* (*ER*) *Model* [1] is a vastly extended conceptual model, and it seems to continue being frequently used, since it is present in the standard *UML* class diagram. This fact justifies the current interest for extending *ER* with behavioural capabilities, that is, procedures which care for integrity constraint enforcement after transaction execution [8].

The research on constraint enforcement in database systems has been developed in many works; in most of them databases have been considered at logical level. Few work has been done on this subject at conceptual level. Some attempts have been made to combine the Entity-Relationship Model with other representation formalisms in order to model dynamic features[10]; in the present work we propose a solution obtained applying an algorithm directly to the *ER* diagram.

When the *ER* Model is used to create a database conceptual schema, a diagram is obtained [3,7]. This diagram, assumed its correction, represents the entity types that constitute the information system and also includes the relationship types between them. Thus, the diagram models the structural features of the system. Each diagram

occurrence represents a snapshot of the system state. The dynamic features that can be modeled are reduced to a set of integrity constraints that limit the valid occurrences of the diagram and then, the allowed transitions of states. The most general constraints are depicted in the *ER* Model by means of different symbols. The minimal update transactions on any diagram object (entity type or relationship type) which are safe with respect to these constraints can be determined from the *ER* diagram independently of the system state [6]. An *update transaction* is a transaction for inserting or deleting an object (the modification of an object is considered in this paper as a deletion followed by an insertion). An update transaction is *safe* if it includes, besides the update operation on the intended object, all other operations on any diagram objects which are required for satisfying the integrity constraints. A safe update transaction is *minimal* if there is not a subset of the transaction which is safe too.

The set of constraints that are considered in this paper are the following:

- The implicit constraint of the *ER* Model which determines that a relationship can exist only if the corresponding participant entities also exist.
- The implicit constraint of the *ER* Model which determines that an entity of an specialized entity type is also an entity of the general entity type.
- The constraints which force the inclusion of an entity of a general entity type into the specialized entity types (total specialization). If this constraint is not required, then specialization is partial.
- The constraint which forbids a general entity type to be included in more than one of the specialized entity types (disjoint specialization). If this constraint is not required, the specialization is said to be overlapped.
- The cardinality constraints which define the minimum and maximum participation of entities in relationships.

The implications that all these constraints have in the transaction design can be analysed directly on the diagram. The set of operations that must constitute an update transaction can be derived from that analysis.

In this paper, an algorithm that obtains the set of minimal and safe transactions with respect to these constraints is proposed. For each entity type and each relationship type of the diagram, and insert transaction schema and a delete transaction schema are generated.

2. An Extended Entity-Relationship Model

The *Extended Entity-Relationship* (*EER*) *Model* used in this paper includes symbols for entity types, relationship types with cardinality constraints and weak entity types [5]. It also includes identification, uniqueness and no null value constraints for the attributes. Finally, it includes generalization/specialization for the entity types and the participation of the relationship types in other relationship types by means of their definition as aggregated entity types. All these symbols can be seen in Figure 1.

Some features of the model are:

(a) The representation of relationship types includes the cardinality constraints. The expression $R(E(1,n), F(0,1))$ in Figure 1 means that each occurrence of E

relates to x occurrences of F ($1 \le x \le n$) through R and that each occurrence of F is related to y occurrences of F ($0 \le y \le 1$). If the minimum cardinality of an entity type E in a relationship type R is equal to 1, then E is said to have the *Existence Constraint* (*EC*) in R; this fact is represented by a double arc connecting E and R.

(b) Weak entity types depend on their participation in one or more relationship types for their identification. They take the identifier attributes from the entities they are related to through those special relationship types.

(c) The specialized entity types are connected to the general entity type through a circle by a line. The specialization properties (*To)tal* or (*P)artial*, and *(D)isjoint* or *(O)verlapped* are specified in the circle.

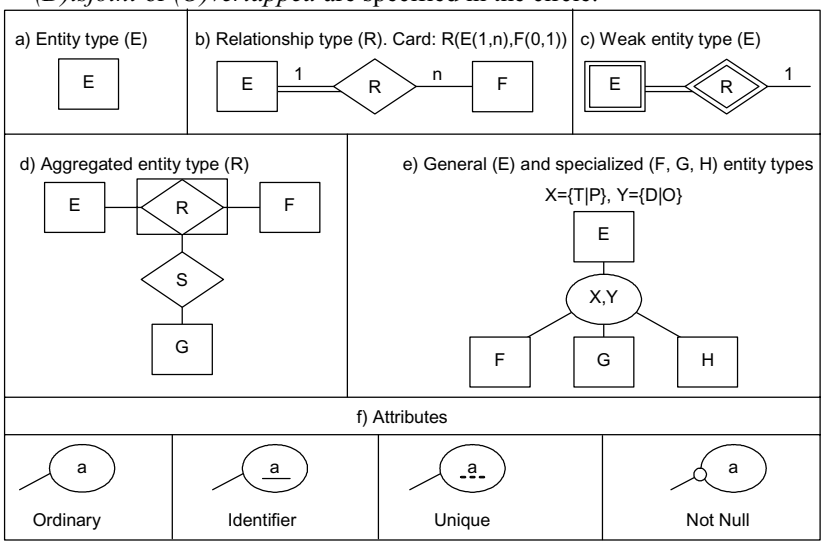


Figure 1. *EER* symbols.

The *EER* Model includes also a transactional language with operators to update objects. An insertion operator (*ins*) and a deletion operator (*del*) are defined for entity types and for relationship types. Also two operators for inserting and deleting specializations are defined. The syntax of these operators is the following:

- `INS_ENT name_ent_type {WHERE condition | ATTRIBUTES attribute_ assignment_list}`
- `INS_REL name_rel_type WHERE condition [ATTRIBUTES attribute_ assignment_list]`
- `INS_SPE name_spe_type WHERE condition [ATTRIBUTES attribute_ assignment_list]`
- `DEL_ENT name_ent_type [WHERE condition]`
- `DEL_REL name_rel_type [WHERE condition]`
- `DEL_SPE name_spe_type [WHERE condition]`

The condition in the previous operations is a formula expressed in a logical language which is similar to the relational tuple calculus; the main difference between these two languages is that in the former there are three kinds of variables: domain, entity and relationship, and a relationship variable can be used to refer one

of its component entities using a dot notation. This formula includes only one free variable which is defined on the type of object which is affected by the operation. The different assignments for this variable that make the formula true correspond to the occurrences that must be inserted or deleted by the operation. In the case of entity insertion, the condition is optional and it would only make sense for insertions from other entity types. In the relationship insertion, this condition is mandatory in order to instance the components of the relationship occurrences that are to be inserted, because it is assumed that, to insert a relationship, the entities that are going to associate in it already exist. In the same way, the insertion of an occurrence in a specialyzed entity type requires that the corresponding occurrence exists in the general entity type. In the deletion operations, the condition is optional; when it is not included, the deletion is applied to all the occurrences of the object type.

3. Approach to the solution

In order to design the algorithm to generate the set of safe update transactions, two kinds of *EER* diagrams can be considered: those without cycles and the ones with cycles. A *cycle* exists if an object type depends, direct or indirectly, on itself for insertion. Initially, only the former are analysed. In section 4.2, the latter are taken into account.

Each integrity constraint, in front of each possible update operation, has different properties with respect to integrity violation and integrity enforcement [9]. These properties determine the way integrity is enforced, if it is possible, when there has been an integrity violation.

Below, the procedure outline is informally described; it will be developed later:

1. Insertion.
   (a) Entity type: if the entity type has the existence constraint in some relationship type, it is also necessary to insert an occurrence in this relationship type.
   (b) Relationship type: if the relationship type participates as aggregated entity with existence constraint in another relationship type, it is necessary to insert an occurrence in this latter relationship type, also.

   The operations that are added to the transaction can require propagation as well. This propagation must be included in the transaction
2. Deletion.

   Deletion operations are defined in two ways: *restrictive* and *in-cascade*. In the restrictive way, as the system controls which constraints are fulfilled, if the deletion is going to leave the database in an inconsistent state, the deletion will not be allowed. In the in-cascade way, apart from the initial deletion operation, those object occurrences which cause an inconsistency are also deleted. The selection of one of these choices must be done with respect to each object type in which the original deleted object participates.
   (a) Entity type:
      i. Restrictive with respect to a relationship type *R*: it only includes the operation of entity deletion.

ii. In-cascade with respect to a relationship type *R*: in addition to the deletion operation of the entity, the deletion of the occurrences from *R* where the deleted entity participates is included in the transaction.

(b) Relationship type

i. Restrictive with respect to an entity or relationship type: it only includes the operation of relationship deletion.

ii. In-cascade with respect to an entity type *E* or to a relationship type *R*: if an entity participating in the relationship to be deleted has the existence constraint in it, then, besides the relationship deletion, it is necessary to include the deletion from *E* for those cases in which the deleted relationship occurrence is the last one in which the entity participates. Furthermore, if the relationship type participates as aggregated entity in another relationship type *R*, besides the previous deletion operations, the deletion of the occurrences from *R* in which the aggregate entity takes part is included in the transaction.

In the in-cascade deletion, the operations that are added to the transaction can require propagation as well. This propagation must be included in the transaction.

In some cases, only the in-cascade deletion is adequate; for example in the relationship $R(E(1,1), F(0,n))$ a restrictive deletion for *E* or *R* is not possible.

4. Transaction generation

In this section, we present a set of algorithms to achieve automatic generation of the minimal and safe update transactions for a given diagram. There are two possible approaches for designing this set of transactions. These approaches are presented below using the example in Figure 2.

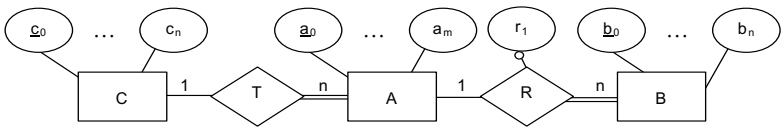


Figure 2: *EER* example diagram

The *modular approach* constructs a transaction for each object type of the diagram (entity type or relationship type) including only one operation (insertion o deletion) and, if it is necessary, calls to other transactions. Therefore, the transactions only take into account the operation on their own objects and the *adjacent* objects that are necessary. Two objects of a diagram are adjacent if they are located at the ends of the same line (in this case, we will say the *distance* between the objects is 1). When the insertion transaction for *B* is designed, a call to the insertion transaction for *R* must also be included in addition to the insertion operation in *B*. Some of the generated transactions are hidden. They are only defined to be called from other transactions, and they are not available to the system users. For example, the insertion transaction for *T* is only used by the insertion transaction in *A*. This is because the cardinality constraints of *A* in *T* (maximum and minimum equal to 1) prevent the *T* insertion transaction from being used alone.

The *complete approach* constructs each transaction by including all the operations that are necessary on any object, regardless of the distance to the initial object in the diagram. Thus, a basic transaction does not include calls to other transactions. In the example, apart from the insertion operation in *B*, the insertion transaction in *B* contains the insertion operation in *R*. Only those transactions that have to be available to the users are generated. In the example, the insertion transaction in *T* is not generated.

The parameters of the transactions are obviously the same in both approaches and they are described as follows:

- In the case of insertion into an entity type, a parameter must appear for each attribute of this entity type. In the case of insertion in a relationship type it is necessary to include a parameter for each identifier attribute of each participant entity type and a parameter for each relationship type attribute. However, if an insertion transaction needs to call other transactions, the parameters of the latter must also be included with the parameters of the former. This problem is solved by the algorithm.
- In the case of deletion, a parameter for each identifier attribute of the object type to delete is needed if it is an entity type. A parameter for each identifier attribute of every component entity type is needed if it is a relationship type.

The generated transaction for the insertion in *A* is presented for both approaches.

```
TRANSACTION Modular_Ins_A (a0x,…,amx,c0x);
   BEGIN
     Ins_ent A attributes a0←a0x,…,am←amx;
     Modular_Ins_T(a0x,c0x);
   END.
TRANSACTION Complete_Ins_A (a0x,…,amx,c0x);
   VAR TX:T;
   BEGIN
     Ins_ent A attributes a0←a0x,…,am←amx;
     Ins_rel T where
          A(TX.A)∧TX.A.a0=a0x ∧ C(TX.C)∧TX.C.c0=c0x;
   END.
```

The in-cascade deletion transaction of the entity type *A* with respect to any object appears below in the modular and complete approaches

```
TRANSACTION Modular_Del_A (a0x);
   VAR AX:A;
   BEGIN
   | Del_ent A where AX.a0=a0x;
   | Modular_Del_T(a0x,-);
   | Modular_Del_R(a0x,-);
   END.
TRANSACTION Complete_Del_A (a0x);
   VAR AX:A, TX: T, RX:R, BX:B;
   BEGIN
   | Del_ent A where AX.a0=a0x;
   | Del_rel T where ¬∃AX(A(AX)∧TX.A=AX);
   | Del_rel R where ¬∃AX(A(AX)∧RX.A=AX);
   | Del_ent B where ¬∃RX(R(RX)∧RX.B=BX);
   END.
```

In the example, on the one hand, the deletion of an entity from the entity type *A* forces the deletion of an occurrence from the relationship type *T*; on the other hand, since the in-cascade deletion has been chosen for relationship type *R*, then it is necessary to delete the occurrences of *R* in which the deleted entity participates. Finally, due to the minimum cardinality constraint of the entity type *B* in *R* (equal to 1), any deletion from *R* determines the deletion of the participant entity from *B*.

A call to a transaction does not require all its parameters to be instanced. Thus, in the call *Modular_Del_R*(a_0x, -), the second parameter is a script; this symbol denotes any value for the parameter that corresponds to it by location in the transaction definition.

The transactions called by the preceding transactions in the modular approach are the following:

```
TRANSACTION Modular_Ins_T (a0x,c0x);
   VAR TX:T;
   BEGIN
   | Ins_rel T where
   |    A(TX.A)∧TX.A.a0=a0x ∧C(TX.C)∧TX.C.c0=c0x;
   END.
TRANSACTION Modular_Del_T (a0x,c0x);
   VAR AX:A;TX:T;S_AX: Set of occurrences of A;
   BEGIN
   | Del_rel T where TX.A.a0=a0x∧TX.C.c0=c0x;
   | S_AX←{AX | ¬∃TX(T(TX)∧TX.A=AX)};
   | FOR EACH AX in S_AX DO
   | | Modular_Del_A(AX.a0);
   | END_FOR;
   END.
TRANSACTION Modular_Del_R(a0x,b0x);
   VAR BX:B;RX:R;S_BX: Set of occurrences of B;
   BEGIN
   | Del_rel R where RX.A.a0=a0x∧RX.B.b0=b0x;
   | S_BX←{BX ¬∃RX(R(RX)∧RX.B=BX)};
   | FOR EACH BX in S_BX DO
   | | Modular_Del_B(BX.b0);
   | END_FOR;
   END.
```

To be able to use the calls to other delete transactions for reaching a consistent system state, it is necessary to include selection conditions to determine which occurrences of other objects must be eliminated. Each one of these conditions is expressed using a formula in the above mentioned logical language. This formula contains a free variable so that all the instances of the free variable that make the formula true are eliminated. All these instances are included in one variable of set type. This set is later crossed, calling to the deletion transaction for each one of the occurrences that are contained in it. The corresponding values of identifier attributes of these instances are used as parameters.

In the deletion transactions for *T* and *R*, there are calls to deletion transactions on other objects of the diagram which have a similar structure. For example, in the transaction *Modular_Del_R*, since entity type *B* has the existence constraint in *R*, when occurrences of *R* are deleted, those occurrences of *B* which no longer participate in *R* must also be deleted. Since the maximum cardinality of *B* in *R* is 1,

we know that for each deleted occurrence of *R*, one occurrence of *B* must be eliminated.

Another important problem relative to the successive calls to different transactions from another deletion transaction needs to be emphasized. When an occurrence of *T* is deleted, there is determined which occurrences of *A* have been left inconsistent (those that do not participate in *T*). This fact prevents falling into an infinite loop of calls, because the deletion of *T* came from a deletion in *A*. Thus, there are no occurrences of *A* that fulfil this condition. Therefore, no call to the deletion transaction of *A* is made, and the possible loop is avoided.

4.1 Algorithm for the transaction generation from an *EER* diagram without cycles

The modular approach has been chosen for the development of the transaction generator algorithm so that the obtained transactions are simpler. The algorithm starts from an *EER* diagram. As a result of the algorithm execution, the set of minimal and safe update transactions which are necessary for the evolution of the system represented by the diagram is obtained. When an insertion transaction calls another insertion transaction, it must be known which parameters must be used. Defining an order in the transaction generation allows that those transactions that are called from another transaction are defined beforehand.

In order to determine the required parameters for an insertion transaction it is necessary to use the basic operation parameters plus all the parameters that occur in each one of the invoked transactions. To avoid undesired duplication in this set of parameters, some of them have to be eliminated: the identifier parameters of the calling object, because they always appear in the newly invoked transaction.

Consequently, it is necessary to design an algorithm that determines the order for transaction generation; this arrangement is also based on the minimum cardinality constraints of the relationship types in which each object participates. The sorting algorithm accepts an *EER* diagram as a unique parameter and it returns an ordered list of all the diagram object names. This algorithm is shown in Appendix A.

As stated above, the arrangement is only necessary to generate the insertion transactions, since in deletion transactions it is not necessary to include parameters from other transactions. The generation of the deletion transactions is done for each object right after the generation of its corresponding insertion transaction. Once the arrangement of the objects of the *EER* diagram is obtained, the transaction generator algorithm is the following:

```
ALGORITHM Transaction_Generator
INPUT   EERX: EER Diagram;
        LI: Sorted list of object names from an EER
            diagram;
OUTPUT  T: Text that consists of the set of safe and
           minimal update transactions for the objects
           in the input EER diagram;
VAR Transaction: Text;
BEGIN
| T ← '';
| FOR EACH object O in LI in order DO
| | IF O is an entity type
```

```
| | THEN
| | | Generate_ins_ent_transaction(O,Transaction);
| | | Join(T,Transaction);
| | | Generate_del_ent_transaction(O,Transaction);
| | | Join((T, Transaction);
| | ELSE /*O is a relationship type*/
| | | Generate_ins_rel_transaction(O,Transaction);
| | | Join(T, Transaction);
| | | Generate_del_rel_transaction(O,Transaction);
| | | Join(T, Transaction);
| | END_IF;
| END_FOR;
END.
```

The algorithms to create safe transactions for the different object types are presented in Appendix B. They are presented in an abbreviated form centered in the calls to other transactions that have to be included to enforce cardinality constraints. The controls that have to be included to avoid inconsistence due to other constraints are left out of this simplified version.

4.2 Algorithm extension to include *EER* diagrams with cycles

When an *EER* diagram contains a cycle, that is, a dependency for insertion, a problem arise when the generated transactions are used.

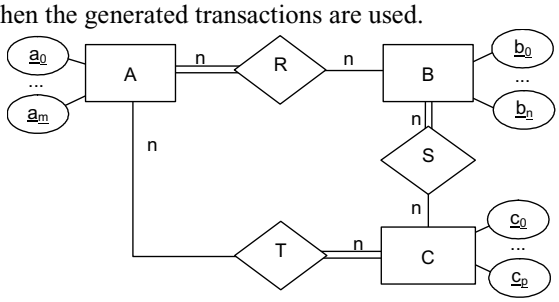


Figure 3: *EER* diagram with a cycle

Effectively, an insertion in *A* requires an existing entity from *B* for inserting a relationship in *R;* but, to insert an entity in B, an existing entity in *C* is needed to insert a relationship in *S*, and this entity also needs an existing entity in *A* to insert a relationship in *T*. This is a deadlock that must be solved by generating, in addition to the normal transactions cited above, special transactions for this situation. The special transactions must include an insertion operation on each object type included in the cycle, plus calls to the modified insertion transactions on these objects. The modified transactions are the normal ones except for that the insertion operation on the intended object type is eliminated. A particular case of cycle is the Total specialization, because the insertion in the general entity type and in one of the specialized entity type must be simultaneous; due to this fact, a special transaction is needed for each specialized entity type, that joins the insertion transaction on the general entity type and the insertion transaction on the corresponding specialized entity type.

Therefore, the defined algorithms for insertion transaction design are modified to take into account the presence of cycles. This extension is not included in the present paper.

5. Conclusions and future work

In this work, we present a new perspective of the Entity-Relationship Model, as a conceptual modelling tool. Traditionally, the *ER* model has been criticized due to its lack of a dynamic dimension. In contrast to this criticism, the authors defend that the integrity constraints defined in a diagram on an Extended Entity-Relationship (*EER*) Model restrict the valid state transitions in the system, and they can therefore determine the minimal transactions that can be performed on it. Following this main idea, given any *EER* diagram we propose an algorithm to obtain, directly from the diagram, the set of safe and minimal transactions with respect to the integrity constraints represented in it. These minimal transactions are the basic units that should be used to design the more general ones that implement the user requirements.

The main features of this work are the following:

1. We propose an extension of the *ER* Model which includes a transactional language. This language includes an insert and a delete operator for entity types, for relationship types and for specialized entity types.
2. We analyse different *EER* diagram patterns in order to develop the algorithm.
3. Finally, we present the algorithm that automatically generates the set of minimal and safe transactions for a given *EER* diagram. The main advantage of the proposal consists of its possible integration into the existing CASE tools based on the *ER* model for conceptual modelling. In this way, these tools could offer the designer a minimal transaction diagram that is able to perform the safe system evolution.

This work can be extended taking into account more general integrity constraints.

Appendix A: Algorithm for sorting the objects of an *EER* diagram

```
ALGORITHM SORT
INPUT EERX: EER Diagram;
OUTPUT L: Sorted list of object names;
VAR Possible: boolean;
BEGIN
| L←empty list;
| FOR EACH object O in EERX DO
| | IF O has not EC in any relationship type
| | THEN
| | | Add(L,O);
| | END_IF;
| END_FOR;
```

```
| REPEAT
| | FOR EACH object O in EERX DO
| | | IF O is not in L
| | | THEN
| | | | Possible←true;
| | | | FOR EACH relationship type R where O has EC DO
| | | | | IF R is not in L
| | | | | THEN Possible←false;
| | | | | END_IF;
| | | | END_FOR;
| | | | IF Possible
| | | | THEN
| | | |   Add(L,O);
| | | | END_IF;
| | | END_IF;
| | END_FOR;
| UNTIL every O is in L;
END.
```

Appendix B: Algorithms for generating the transactions

```
PROCEDURE Generate_ins_ent_transaction
INPUT   E: text /* Entity type name */;
OUTPUT  T: text /*Transaction*/;
VAR Trans_Parameters, Operations: Text;
BEGIN
| Add to Trans_Parameters a parameter for each
|    attribute of E;
| Generate the insertion operation into the entity
|    type E;
| Add this operation to Operations;
| FOR EACH relationship type R where E has EC DO
| | Add to Trans_Parameters a parameter for each
| |      parameter of the insertion transaction into R |
| |      that is not yet in Trans_Parameters;
| | Generate a call to the insertion transaction into R;
| | Add this call to Operations;
| END_FOR;
| T←'TRANSACTION';
| Join(T, 'Ins_Ent_Transaction_', name(E),
|     Trans_Parameters,'BEGIN', Operations, 'END.';)
END.
PROCEDURE Generate_ins_rel_transaction
INPUT           R: text /*Relationship type name */;
OUTPUT  T: text /* Transaction*/;
VAR  Variables, Trans_Parameters, Operations: Text;
BEGIN
| FOR EACH entity type E that participates in R DO
| | Add to Trans_Parameters a parameter for each
| |     identifier of E;
| END_FOR;
```

```
| Add to Trans_Parameters a parameter for each attribute
|      of R;
| Add to Variables a variable defined over R;
| Generate the insertion operation in the relationship
|      type R;
| Add the operation to Operations;
| FOR EACH relationship S where R has EC as aggregated
| |      object DO
| | Add to Trans_Parameters a parameter for each
| |       parameter of the insertion transaction into S
| |       that is not yet in Trans_Parameters;
| | Generate a call to the insertion transaction into S;
| | Add this call to Operations;
| END_FOR;
| T←'TRANSACTION';
| Join(T, 'Ins_Rel_Transaction_', name(R),
|      Trans_Parameters, 'Variables', Variables, 'BEGIN',
|      Operations, 'END.');
END.
```

PROCEDURE Generate_del_ent_transaction

```
INPUT   E: text /* Entity type name */;
OUTPUT  T: text /*Transaction*/;
VAR Variables, Trans_Parameters, Operations: Text;
BEGIN
| Add to Trans_Parameters a parameter for each identifier
of E;
| Generate a variable on E;
| Add this variable to Variables;
| Generate the deletion operation from the entity type E;
| Add this operation to Operations;
| FOR EACH relationship type R where E participates DO
| | IF the deletion is in-cascade with respect to R
| | THEN
| | | Generate a call to the deletion transaction from R;
| | | Add this call to Operations;
| | END_IF;
| END_FOR;
| T←'TRANSACTION';
| Join(T, 'Del_Ent_Transaction_', name(E),
|      Trans_Parameters, 'Variables', Variables, 'BEGIN',
|      Operations, 'END.');
END.
```

PROCEDURE Generate_del_rel_transaction

```
INPUT   E: text /* Relationship type name */;
OUTPUT  T: text /*Transaction*/;
VAR  Trans_Parameters, Operations, Variables: Text;
BEGIN
| FOR EACH entity type E that participates in R DO
| | Add to Trans_Parameters a parameter for each
| |       identifier of E
| END_FOR;
| Generate a variable on R;
| Add this variable to Variables;
```

```
| Generate the deletion operation for R;
| Add this operation to Operations;
| FOR EACH relationship type S where R participates
| |          (*as aggregated object*) DO
| | IF the deletion is in-cascade with respect to S
| | THEN
| | | Generate a call to the deletion transaction for S;
| | | Add this call to Operations;
| | END_IF;
| END_FOR;
| FOR EACH entity type E that participates in R with EC
| |     DO
| | Generate a set variable on E;
| | Add this variable to Variables;
| | Generate a query on this variable;
| | Generate a FOR loop on this variable including
| |     a call to the deletion transaction of E;
| | Add this loop to Operations;
| END_FOR;
| | T←'TRANSACTION';
| Join(T, 'Del_rel_transaction', name(R),
|      Trans_Parameters, 'Variables ', Variables, 'BEGIN',
|     Operations, 'END.')
END.
```

References

[1] P.P. Chen. The Entity-Relationship Model: Towards a Unified View of Data. ACM TODS, Vol. 1, No. 1, pp. 9-36, 1976.

[2] G. Engels, M. Gogolla, U. Hohenstein, K. Hülsmann, P. Löhr-Richter, G. Saake, H. D. Ehrich. Conceptual Modeling of Database Applications Using an Extended ER Model. Data and Knowledge Engineering, Vol. 9, No. 2, pp. 157-204, North Holland, 1992.

[3] R. Elmasri, S.B. Navathe. Fundamentals of Database Systems. Third Edition. Addison-Wesley, 2000.

[4] M. Gogolla. An Extended Entity-Relationship Model. Fundamentals and Pragmatics. LNCS 767, Springer-Verlag, 1994.

[5] L. Mota-Herranz. M.A. Pastor. Diseño conceptual con el modelo Entidad-Relación. Departamento de Sistemas Informáticos y Computación, Internal Report, DSICID/56/97, 1997.

[6] J.A. Pastor-Collado. Supporting Transaction Design in IS Conceptual Modelling through Pre-synthesised Update Transaction Specifications. EXSELSI'95. 1995.

[7] B. Thalheim. Entity-Relationship Modeling. Foundation of Database Technology. Springer-Verlag, 2000.

[8] Balaban, M. and Shoval, P. MEER -An EER Model enhanced with structure methods, Information Systems, 27 (245-275), 2002.

[9] Pastor, M. A., Celma-Giménez, M., Mota-Herranz, L. Análisis de Restricciones de Integridad en el Nivel Conceptual. Accepted in IDEAS'04.

[10] C.A. Heuser, E.M. Peres, G. Richter: Towards a complete conceptual model: Petri nets and EntityRelationship diagrams, Information Systems, Vol. 18, No. 5, 1993, 275-298

DB APPLICATIONS

SCIENTIFIC PAPERS UNIVERSITY OF LATVIA. 2004. VOLUME 672, Datbases and Information Systems
125–133 P.

An Output Schema for Multimedia Data in Multimedia Database Systems

Thomas Heimrich

Technical University of Ilmenau,
Databases and Information Systems,
D-98684 Ilmenau
thomas.heimrich@tu-ilmenau.de

Abstract. Multimedia data differ significantly from alphanumeric data. The semantic of multimedia data depends from the data presentation. Up to now it is only possible to model structure and behaviour of multimedia data in a schema of a multimedia database. This paper proposes a new concept, called *output schema*. The output schema can be integrated into the multimedia database schema. An output schema is a description of the multimedia data output. It can be reused easily and adapted. The multimedia database can use the information from the output schema to optimize the data output and the data storage.

Keywords: multimedia databases, modelling data output

1. Introduction

Multimedia database management systems are mostly not completely new developed systems. Usually already existing object-relational or object-oriented database systems are used to store multimedia data. A multimedia layer is built on its top.

A multimedia database must support different multimedia types (e.g. image, video, audio, fulltext) as basis data types. The database must offer also efficient storage structures and indices for these media types. Some extensions of commercial databases (e.g. DB2, ORACLE) offer these capabilities.

It is possible to build complex multimedia documents from these basis media types. The complex multimedia types are defined in the structure and behaviour schema of a multimedia database. The structure schema consists of different structure types. A subtype-relation is defined on these structure types. With that a IS-A-Hierarchy can be build by means of inheritance. The possibilities of the inheritance concept are very useful for modelling complex multimedia data.

The behaviour schema defines a set of methods. These are assigned to structure types. Usually it is desirable for behaviour inheritance to process a IS-A-Hierarchy of types. In this case it is necessary to allow that only methods in subtypes may be specialized, that is overloading is only allowed with compatible signatures (contravariance).

The data output is very important for the semantic of the multimedia data. The semantic of an audio for example gets lost if its presentation speed is too fast. Up to

now it is only possible to model structure and behaviour of complex multimedia documents in a type system. We introduce the new concept of an output schema. It is supposed to model relationships during the data output. It is a description of the data presentation. Instances of this description are specific presentations.

Section 2 gives a survey over some related work. Section 3 introduces the output schema and its requirements. Section 4 gives a notion about how the output schema may be implemented. Section 5 summarizes and concludes this paper.

2. Related work

The need for modelling multimedia data representation is known since computer can handle multimedia data [5]. One of the most popular presentation languages is SMIL [8]. The multimedia data used by a SMIL presentation are stored as files in a filesystem. The filename is used to refer to multimedia data in a SMIL-Script. A SMIL presentation is inflexible. It is for example not possible to use the same SMIL presentation for picture *Image_A* and later for picture *Image_B*. The reuse of the structure defined by the SMIL-Script is not possible. If data are changed there is no way to announce these changes automatically to the SMIL-Script. Thus the SMIL-Script can not adapt its definitions according to these changes. The changes do not take into account what is defined by the SMIL-Script.

An algebra for creating and querying multimedia presentations is described in [1]. The multimedia presentation algebra (MPA) contains generalisations of select, project and join operations in the relational algebra. This algebra operates on trees whose branches reflect different possible presentations of a presentation description. *Creating a presentation* means to create an instance of a presentation description. How to describe a presentation is not part of that paper.

Candan et. all [6] have developed a view management for multimedia databases. They introduced virtual objects with associated spatial and temporal presentation constraints. Materializing a dynamic multimedia view corresponds to assembling and delivering an interactive multimedia presentation according to the visualization specification. The paper describes mainly the personalisation of multimedia views. Boll et. all [3, 4] have developed the ZyX data model for multimedia documents. It is a presentation-neutral description of a multimedia presentation. A presentation in the ZyX model is also a tree of specific presentation nodes. This model was implemented with a object-relational database. Classes were developed to model the presentation nodes. An disadvantage of this model is the small support for temporal relationships between multimedia data. Only the parallel and sequential presentation of media objects is supported. If an object model is used then it is impossible for the database to optimize the data output.

Up to now a presentation description is not part of the multimedia database schema. Thus the database has no knowledge about the desired data output streams and the output conditions. It can not optimize the multimedia data output nor ensure the consistency between the presentation description an the stored multimedia data.

3. Output schema

Section 3.1 defines some requirements which an output schema should fulfill. Then a formal definition for an output schema is proposed. Next the inheritance for output types is introduced.

3.1. Requirements on an output schema

The output of multimedia data is more complex than the output of alphanumeric data. There are synchronisation relationships that have to be taken into account. During the output a certain quality of the data has to be guaranteed. Otherwise multimedia data might be semantically falsified.

Up to now multimedia database systems were often used for the storage of multimedia data independent of each other. The data themselves are stored but only very few relationships between these data. It is assumed that the user builds the relations between the multimedia data through a database query. A query could be: "Search Video1 and Audio1 and output them parallel." In [7] a formal query language is proposed for that. *Constrained queries* can be built. In these queries it is possible to specify temporal and spatial relationships (e.g. start(video2) ≤ end(video1) + 0.01) for the data presentation. The original query will be enlarged with a conjunction of these constraints. A corresponding enlargement of SQL seems to be easy to realize. But the following aspects are problematic:

- The user has to describe a complex output in the query.
- The specification of the output can only be used once. If the same output is desired again, the whole specification has to be built again.
- The user has to determine the temporal/spatial semantic of the output.
- There are no synchronisation relationships stored in the database. Thus the semantic of the multimedia data may be lost.

We believe, that the presentation descriptions must be part of the multimedia database schema. The reasons for that are:

- Multimedia documents consist of multimedia data and complex spatial/temporal relationships between them. Without storing a presentation description it is not feasible to store and restore multimedia documents.
- The presentation of multimedia data is very important for their semantic. The resolution of images for example has high influence on their semantics. It has to be specified for the data output. Otherwise it may occur that the user sees the shown image with a too low resolution. Thus he can not see all the information of the image and he gets semantically incomplete or wrong data.
- The database needs the information about the desired presentation of the data in order to optimise the data output and the used data organisation. For example a specific video shall be presented parallel to a specific audio. Now it is important that the database chooses a storage organisation that allows the parallel output of the video and audio.

It is necessary to model the data presentation in a multimedia database. The model must have the following criteria (q.v. [3]):

Reusability. Reusability of presentation description should be supported along two dimensions. Firstly reusability means that a presentation description can consist of other presentation descriptions. Secondly it means that from a presentation description many presentations can be built. A presentation description is a template for presentations.

Adaptation. Extensions to and chances of presentation descriptions should be easy to do. Users should be able to adapt existing presentations according to their needs.

Presentation-neutral Representation. The presentation description must be independent of the format used by the presentation. Therefore a multimedia database can support different presentation formats. The presentation-neutral description has to be converted into a presentation-specific format which is used for the playout of the multimedia data. It is desirable that this conversion is lossless.

3.2. Formal definition of the output schema

Now the concept of the *output schema* is introduced. That is how model the data output of multimedia data. Output types are defined. They can be used as a reusable, adaptable and presentation-neutral representation for the data output.

The output type is created by the user once and thereafter managed by the database. When querying the database, a particular output type can be used. It is not necessary to describe a complex output in the query. The output type is reusable and thus can be used for the optimization of the data output. A formal definition of a output type system (*OT*) is given as follows:

(i) video, audio, image, fulltext $\subseteq OT$

(ii) If ot1, ot2 $\in \{OT - \{image, fulltext\}\}$, TC={before, meets, overlaps, during, starts with, finishes with, equals} and $tc \in TC$ then ot1 *tc* ot2 $\in OT$

(iii) If ot1,ot2 $\in \{OT - audio\}$, SC={left, right, over, under} and $sc \in SC$ then ot1 *sc* ot2 $\in OT$

(iv) If ot1,ot2 $\in OT$, then ot1 *substitute* ot2 $\in OT$

(v) if ON_i are different names for output types and $ot_i \in OT, 1 \le i \le n$, then $[ON_1 : ot_1, ..., ON_n : ot_n] \in OT$ (tuple of output types)

(vi) UDOT $\subseteq OT$, UDOT is a set of names for user defined output types.

The basis output types are defined in (i). These are the basis multimedia types which should be supported by a multimedia database. That is why these multimedia types should have a default presentation.

Temporal output constraints are defined in (ii). The Allen-Relations [2] are used. It is also possible to use other concepts concerning temporal relations, for example *difference constraints* The important aspect here is that by means of combination any complex output type can be created. Temporal relationships can be defined only

on multimedia types which have an temporal dimension. *Image* and *fulltext* do not have this dimension. Thus it is not possible to define an output type with temporal relationships for these multimedia types.

Spatial constraints are defined in (iii). Thus it is possible to specify also spatial relationships between multimedia data (video, image, fulltext) in an output type system. It is not possible to define spatial relationships for the multimedia type *audio.* This type has no spatial dimension.

With (ii) and (iii) the synchronisation relationship between multimedia objects is modelled. Synchronisation is a special relationship that exist only between multimedia objects. In many cases the output of media objects should take place in a specific temporal and/or spatial order. Synchronisation relationships define the temporal/spatial data flow of the output.

Another relationship between multimedia data is the *substitution relationship*. The same information can be represented by different media (e.g. speech or text). The designer of a media object decides which kind of representation he prefers (for example speech). He can allow another form of presentation as an alternative (for example text). The substitution relationship is defined in (iv).

Point (v) defines a tuple type for output types. Thus it is possible to define smaller parts of a complex output type and give them a name. Through these names these parts can be used within the complex output type definition. An example for a tuple type is: [c1: *video1 before video2*, c2: *video3 after video4*]. The tuple type defines an array (tuple) of constraints. Every constraint has a name. It is assumed that a presentation build from a tuple output type has to satisfy all constraints which are defined in the tuple type.

User defined output types (UDOTs) are defined in (vi). A user defined output type can be called from other output types through its name. We can use these types for building complex output types.

An output type defines a template for a set of specific presentations. Assume the definition of the following UDOT: *myUDOT: video equals audio*. The output type has the name *myUDOT*. Through this name the output type can be used e.g. in queries. Here the output type defines the temporal ordering of multimedia data, that is a video and an audio have to be presented equally. Every presentation that presents a video and an audio equally is an instance of this output type. The output type can be reused with different specific multimedia objects. We used a presentation-neutral representation for output types. This representation can be transformed into different presentation languages. Through that the multimedia database can offer format independent presentations.

3.3. Inheritance for output types

It should be possible to order output types in a output-type-hierarchy (IS-A-relationship). By that output types can be specialized. Existing output types can be easily re-used and adopted. An output type that outputs “ot1 before ot2” can be specialized to “ot1 two seconds before ot2”. By means of derivation also substitutions between multimedia data can be supported. The output type “video equals audio” could be specialized to “video equals fulltext”. This output subtype

allows the parallel output of video and audio and as an alternative the parallel output of video and fulltext.

It is possible to build output subtypes from all output types by means of derivation. A output type o*t'* is output subtype of output type o*t* if every instance of o*t'* is also an instance of o*t*. An instance of an output type is a specific presentation. For example an output type is specified as "video equals audio". Every presentation in which a specific video is played out equal to a specific audio is an instance of that output type.

We introduced a relation „$\leq$" for output subtypes. For example the output type "video two seconds before audio" is an output subtype from the output type "video before audio". All instances from the output subtype are also instances from its parent type. An output subtype has stronger output conditions than its parent type. The number of instances of an output subtype is smaller than those of its parent output type.

The subtype-relation „$\leq$", $\leq \subseteq OT \times OT$ is defined as follows:

(i) $ot \leq ot$ for every $ot \in OT$

(ii) $[OT1' : ot1', ..., OTm' : otm'] \leq [OT1 : ot1, ..., OTn : otn]$ if

(a) $m \geq n$ and

(b) $(\forall OT_i, 1 \leq i \leq n)(\exists OT_j', 1 \leq j \leq m) OT_i = OT_j' \wedge$

$(ot_j' \leq ot_i \vee ot_i$ *substitute* $ot_j')$

The subtype relation for tuple output types is defined in (ii). It is assumed that a presentation to a tuple output type must hold all defined constraints Thus point (ii)(a) defines that an output subtype can have more output constraints than its parent output type. For example the output type [c1: *video1 before video2*, c2: *video3 after video4, c3: video1 equal audio1*] can be a subtype of [c1: *video1 before video2*, c2: *video3 after video4*]. The output types defined in the output subtype have to be more specific than the output types defined in the parent output type. This is defined with (ii)(b). The output type [c1: *video1 3 seconds before video2*, c2: *video3 after video4*] can be a subtype of [c1: *video1 before video2*, c2: *video3 after video4*]. This example also shows why the conditions in a tuple output type must have names. Only through these names it is possible to determine whether a constraint is more specific than its definition in the parent output type.

Through the inheritance of output types it is easy to adapt existing output types. Existing output types can be easily reused.

4. Implementation

4.1. Definition of output constraints

In order to model multimedia data it is necessary to model their structure, behaviour and presentation. A concrete class definition must consider all these parts. An example for a class definition is:

```
Class example{
// structure type
        NameOfVideo video;
        NameOfAudio audio;
// behaviour type
        setNameOfAudio(a audio);
// output type
        NameOfVideo equal NameOfAudio
}
```

To keep it simple we used pseudo code for this example. To create a specific database schema a data manipulation language has to be used.

The novelty here is, that a class describes its own presentation and that the output types of classes can build a IS-A-Hierarchy (section 3.3). Output schema and structure schema are very similar. The structure schema in a object-relational databases defines the schema of typed tables. The output schema defines the presentation of all entities in these typed tables. If an user asks for entities from that table the database knows how to present the result. The database designer can determine how multimedia data should be presented.

The output schema is defined in a very abstract way. In a concrete implementation a component in the database is needed which converts these presentation-neutral representations into a specific presentation language (e.g. SMIL). The user sends a query to the multimedia database and gets e.g. a SMIL-script and the required data as a result. A presentation client (e.g. RealPlayer) is used to show the multimedia presentation what represents the final result of the database query.

4.2. Checking the output constrains

The shown way to define output constraints is very simple for the database designer. It is very hard to check these constraints in that form. From that we transform the Allen-Relations into *different constraints*. As an example the constraint *video equal audio* can be written with different constraints as follows:

$$start(audio) - start(video) \leq 0\,,\ start(video) - start(audio) \leq 0$$

$$end(audio) - end(video) \leq 0\,,\ end(video) - end(audio) \leq 0$$

It is easy to see that a large set of different constraints can caused by some simple Allen-Relation. A constraints graph can be build from different constraints. With a constraint graph the set of different constraints can be resolved in polynomial time. There is no conflict in the defined set of different constraints if the constraint set is solvable. Furthermore the result for the constraints set can be seen as a schedule for the data output.

Different constraints can easy be checked during the data output. The concrete time points for the start and the end of media objects can be used in the different constraints.

4.3. Using an output schema

Often users do not ask for complete objects from a specific class. In ad-hoc queries users ask for attributes from different tables. A presentation description for that kind of query is necessary. It is possible to define output types as user defined output types (UDOT). Those are independent from the structure and behaviour schema. The user can define a output schema in the way it is shown in section 3. A user can build arbitrary output types in the database. In a query these output types can be used instead of describing the complete presentation in the query. It is important that the attributes which should be presented must match the parameters of the used output type. An example is:

```
SELECT video1, audio1
FROM tableName
WITH OUTPUT TYPE myUDOT(video1,audio1)
```

Thus it is possible that a user defines its own output types. He has not to use the output types which are defined by the database designer.

Still a multimedia database must also have the capability to describe the complete presentation in the query. Nobody is able to predict every possible kind of the presentation at modelling time.

5. Conclusion

When looking at multimedia data it is not automatically known how to present these data correctly. Thus a model for the structure, the behaviour and the output is needed. This paper introduces the concept of the *output schema*. Thus it is possible describe the presentation of multimedia data. A subtype relationship for output types was also defined.. So a IS-A-Hierarchy for output types can be build. The output types are a reusable, adaptable and presentation-neutral representation for the data output. A output type can be part of a class definition. Thus the developer of the class can define the presentation of the multimedia data. A output type can also be independent from structure and behaviour schema. A user can define its own output types. This kind of output types can be used within a database query. Thus a definition of complex presentation constrains in a query is not necessary anymore.

A big advantage of the proposed output schema is its integration in the database schema. Through that a multimedia database can optimize the data output and the used data structures. Furthermore it is easy to see that presentation constraints can be checked by the database. If an attribute from type video e.g. changes its length then it is possible to check all output types for that attribute. The database can determine the invalid output types. The concept of the output schema is very similar to structure and behaviour schema. Thus it is easy to integrate the output schema into existing object-relational or object-oriented databases.

References

[1] Adali S., M.L. Sapino and V.S. Subrahmanian: *An algebra for creating and querying multimedia presentations.* Multimedia Systems, 8:212-230, 2000

[2] Allen J.F.: *Maintaining Knowledge about Temporal Intervals.* Communications of the ACM, 26(11):832-843, 1983

[3] Boll S. and W. Klas: ZyX - A Semantic Model for Multimedia Documents and Presentations. In: Semantc Issues in Multimedia Systems, pages 189–209, 1999.

[4] Boll S. and W. Klas: ZyX – A Multimedia Document Model for Reuse and Adaption of Multimedia Content. TKDE 13(3): 361-382, 2001

[5] Christodoulakis S. and Analyti A.: *Guest Editorial Special Issue on Multimedia Information Systems.* IS 20(6): 443-444 (1995)

[6] Kasim S. C., E. Lemar and V.S. Subrahmanian: *View Management in multimedia databases.* The VLDB Journal, 9(2): 131-153, 2000

[7] Marcus S. and V.S. Subrahmanian: *Foundations of Multimedia Database Systems.* Journal of the ACM, 43(3): 474-523, 1996

[8] W3C: *Synchronized Multimedia Integration Language (SMIL 2.0).* http://www.w3.org/TR/smil20, 2001

SCIENTIFIC PAPERS UNIVERSITY OF LATVIA. 2004. VOLUME 672, Datbases and Information Systems
134–144 P.

Concurrent Video: Operational Extensions[1]

Oleg Proskurnin

University of St.Petersburg, Russia
olegpro@acm.org

Abstract. Today, non-linear video editing technology became ubiquitous, having spread from the film industry to almost every home computer. Dozens of multimedia authoring applications offer a wide range of functional capabilities, yet there is an evident lack of support for collaborative activities among them. To change this situation, a novel data model called concurrent video has been recently proposed. Since it was substantially based on formal aspects of cooperative transactions, it has provided a solid foundation for consistent exchange and sharing of information between co-workers in collaborative environments. The main objective of this work is to extend the concurrent video model by introducing advanced editing operations, which should permit more flexibility within the authoring process and enable a higher level of concurrency among users' actions. Besides, this extension is going to be performed in such a way that major transactional properties of the original proposal will be preserved.

Keywords. Concurrent video, video modeling, collaborative multimedia authoring, CSCW, cooperative transactions, consistency, CoAct

1. Introduction

It is a well known fact that the possibility to collaborate in multimedia authoring environments is going to bring considerable advantages into the process of content creation. With reasons ranging from tight project deadlines to distribution of tasks among editors, support for joint activities could noticeably improve video post production technology in all related industries [15].

Recently, a technique for collaborative authoring of media streams, such as video or audio, has been presented in the form of a concurrent video data model [11]. Based on the cooperative activity framework CoAct [13, 18, 9], this model has enabled consistent sharing of information in the face of concurrent modifications, which is often regarded as the key task of multi-user applications design.

The background is that co-authors' actions in the CoAct framework are represented via activity histories, which are basically sequences of editing operations constituting individual working processes. As a result, consistency of information exchange is achieved by means of the history merging mechanism [18] that takes into account compatibility relations of participants' actions in order to detect conflicts in their joint efforts.

[1] This work is partially supported by the Russian Foundation for Basic Research under grant No. 04-01-00173

To avoid unnecessary conflicts, editing operations offered within the concurrent video model were especially designed to have good commutativity properties. In fact, according to the concepts of timeline-based video authoring [4], these operations allowed participants to work concurrently on different clips, which in the end formed the desired video material.

The goal of this paper is to improve the original data model by providing support for non-conflicting cooperative activities within the same clips. However, most transactional aspects of the previous proposal are going to be kept intact.

The rest of this paper is organized as follows. After giving a brief overview of the related work, the improved version of the data model is introduced. Next, transactional properties of concurrent video are discussed and finally conclusions and further work are presented.

2. Related Work

This section first gives a brief review of the CoAct framework and then presents a summary of previous work on video modeling and collaborative multimedia authoring.

2.1. The Cooperative Activity Model

The cooperative activity framework CoAct [13] defines a generic transactional model that enables consistent sharing and exchange of information in various collaborative scenarios, like multi-user document authoring environments [14].

To support joint activities, the CoAct framework assigns a personal isolated workspace containing copies of shared data to each user involved in the cooperative effort. Also, a separate common workspace is established to represent the current state of team work. Thus, participants are able to manipulate their private versions of shared objects independently from each other, while the common database stores at first initial data, then intermediate and lastly final results of the joint effort.

At the user level, cooperation is achieved by explicit exchange of the contents of workspaces, which can be performed either directly or through the common database by means of dedicated operations. But conceptually, these processes are controlled by the history merging mechanism [18, 9], that takes into consideration semantics of the application domain, ensuring correctness of information exchange.

The point is that participant's actions in CoAct are modeled by an activity history, which is basically a sequence of operations carried out by a certain user in his workspace. Similar to conventional concurrency control [20], a semantic conflict test is defined for each possible pair of operation invocations. Such compatibility predicate is often symmetric and is typically specified in terms of state-independent commutativity relations. Since the order in which commuting operations are executed is irrelevant, the compatibility property is used in CoAct as a basis for merging activity histories. In other words, two operation invocations coming from different users are both allowed to be present in the merge only if they are compatible. Otherwise, the controlling user has to resolve the conflict by discarding unnecessary actions.

Consequently, the merging mechanism of the CoAct model enables semantically consistent incorporation of individual activity histories into a single one that is implicitly formed in the common database and represents the results of cooperative work.

2.2. Collaborative Video Authoring

Although a variety of multimedia authoring tools are available in the market, e.g. [2, 16], none of them supports concurrent video editing and the related research proposals are also very few.

Some of these works investigate cooperative activities based on locking mechanisms [3, 17], while other approaches consider collaboration in real-time groupware systems only [22]. In contrast to the concurrent video model [11], these proposals do not cover multimedia modeling issues and do not provide consistency guarantees in a transactional sense.

However, there exists a wealth of research works examining various aspects of video data management and some of them have common features with the presented investigations.

In particular, operations on media segments defined in video and stream algebra [21, 10], as well as updating mechanisms in other proposals [1, 6] partially resemble editing facilities of the model described below. Moreover, previously introduced hierarchical data structures [1, 21] and video objects [12] also have certain correlations with the presented approach.

Nevertheless, the major distinction between concurrent video and former models lies in the suitability of the new proposal for the basic needs of both video authoring systems and cooperative environments, which has not been achieved ever before.

The point is that in the considered authoring systems the movie is created by altering and assembling source clips which are treated as independent units of raw material. These clips along with other media objects are placed on the timeline, which naturally represents the time flow. After the inclusion, objects can be edited, special effects and transitions can be applied and the resulting material can be arranged into a final video production. In addition, since such editing implies just referencing the original files, which are not actually modified, a special process called rendering is required to preview or export the produced movie.

Concurrent video takes into account these basic concepts of timeline-based authoring and provides an efficient tree-based abstraction for referencing a sequence of media clips. It also presents a set of high-level non-destructive editing operations allowing users to insert and delete, alter and temporally combine video material to form the final production.

To support joint activities, operations on different clips are considered to be commutative, i.e. different users can work concurrently on different clips without any conflicts. Moreover, hierarchical structure that references video data also stores activity histories, providing elegant support for cooperation mechanisms of CoAct.

The above features clearly distinguish concurrent video model [11] from other research proposals in the area of cooperative authoring. In particular, this investigation significantly differs from recent works [8] and [5], which present tree-based data structures for synchronous collaborative document editing.

3. Enhanced Concurrent Video Model

This section outlines improved concurrent video model that provides advanced intra-clip editing operations and is entirely based on its already presented counterpart [11].

3.1. Video Segments

In the considered model video segments represent an abstraction over independent units of video data, which are used as building blocks within the authoring process.

Basically, each video segment references a contiguous part of raw material via a frame sequence and has its own set of attribute-value pairs which describe the proper interpretation of the underlying media at the presentation or rendering level. Frame sequences reflect the stream-like nature of video data, while attributes support implementation of non-destructive editing operations as well as specifications of desired transitions and special effects for the segment.

Actually, both the definition and the purpose of video segments in this work stay the same as in the original proposal [11], however, we quote some facts here for convenience:

Definition 1 (frame sequence) A frame sequence is a finite non-empty sequence $(f_1, \ldots, f_N)$ of video frames f_i, referring to a contiguous block of video data. N is called the length of a video sequence F and is denoted by $Length(F)$.

Definition 2 (video segment) A video segment is a pair (F, A), where F is a frame sequence and A is a possibly empty set of attribute-value pairs $\{a_i{:}v_i\}$, storing various additional information about the containing segment.

Besides, two trivial operations – concatenation and extraction – can be defined for frame sequences [11], however they won't be referred in this work explicitly and thus are omitted.

3.2. Video Activity Tree

Concurrent video employs a single hierarchical data structure, called video activity tree, for modeling both underlying media and cooperative transactions.

Basically, the structure of an activity tree in this work remains the same as before: leaf nodes, called valid, are associated with video clips that authors place on a timeline, dead nodes stand for previously deleted clips and the rest nodes hold related parts of the activity history. Also, activity history elements are marked whether they are private or shared to indicate what part of the tree is present in the common database and what part exists only in the considered local workspace, as illustrated in figure 1.

However, unique identifiers assigned to video clips, which have enabled the development of editing operations with state-independent commutativity relations in [11], are not sufficient for the purposes of this work. In order to provide intra-clip operations with similar commutativity properties, unique values should be assigned to all frames constituting the video stream. Fortunately, such task can be performed

by combining clips' identifiers with frames' indices within these clips. This approach is actively exploited by novel editing operations and is reflected below in the redefinitions of activity tree's node structures.

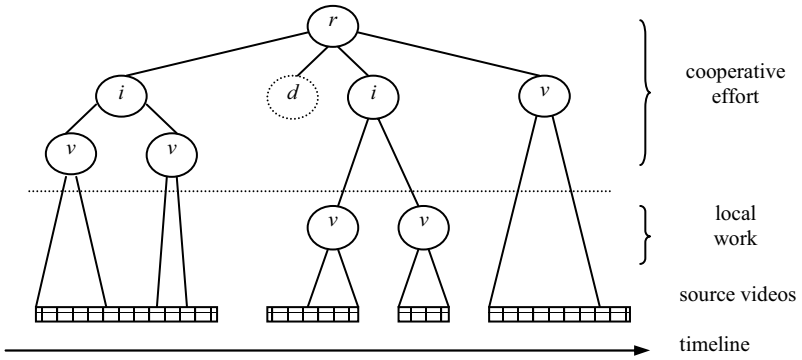


Figure 1. A sample video activity tree in a user's workspace. Characters *v*, *d*, *i* and *r* denote valid, dead, intermediate nodes and the root node of the tree respectively.

Definition 3 (video activity tree) A video activity tree T is a tuple (*Root*, *Nodes*, *Valid*, *Dead*), where *Root* is the root node of T, *Nodes* is a set of intermediate nodes of T, and *Valid* and *Dead* are disjoint sets of so-called valid and dead leaf nodes respectively.

Additionally, there exists a total order $<_t$ defined over the set of leaf nodes of T, that corresponds to the order in which associated video clips are (or were for dead nodes) located on the timeline.

Definition 4 (valid node) A valid node V of a video activity tree is a tuple (*NID*, *Range, Segment*, *History*), where *NID-Range* pair uniquely identifies V's associated video data, *Segment* is the video segment representing this data and *History* is an ordered set of operation instances related to the given node V. *Range* is represented as a pair of bounding values [*Low*, *High*].

Definition 5 (intermediate node and dead node) An intermediate node as well as a dead node of a video activity tree is merely a tuple (*NID*, *Range, History*), whose elements are intended for the same purposes as their counterparts in valid nodes (except *NID-Range* pair identifies no data).

Operation instances, mentioned above, can be treated as records describing the fact of execution of a certain editing operation by some user. They act as elementary entities for building activity histories and modeling the authoring process, being the same as in the original work [11]:

Definition 6 (operation instance) An operation instance is a tuple (*Status*, *OID*, *Name*, *Input*, *Output*), where $Status \in \{private, shared\}$ indicates whether this instance is present only in the current workspace or not, *OID* is a unique identifier of this instance (required for tracing identical instances during the merging process) and *Name* is the name of the executed operation, whose input and output parameters are reflected in *Input* and *Output* sets.

3.3. Editing Operations

The main challenge in the design of editing operations is to provide a solid support for modeling authors' actions in a collaborative environment. Thus, it is worth to remember that the most common thing cooperating authors are likely to do is altering and assembling video clips on a timeline, and what is more, they are likely to do it concurrently.

In contrast with the original proposal [11], which has provided only segment-level editing operations, in this work most attention is focused on frame-level manipulations and achieving commutativity of semantically non-conflicting actions performed within the same video segment, e.g. altering of different frames of the same clip by different users.

First of all, an initialization routine intended for fixing an initial state of the cooperative activity in the common database is provided.

Definition 7 (initialization algorithm) The initialization algorithm takes a sequence of N video clips, denoted by a collection of N video segments $(vs_1, \ldots, vs_N)$ as input and forms a corresponding video activity tree with an empty history:

1. Create N valid nodes V_i: $\forall\, i,\ 1 \le i \le N$: $V_i := (NewID(), [1, Length(vs_i.F)], vs_i, \varnothing)$,
   where $NewID()$ is a function generating a new unique identifier,
   see [11] for how it may work.
2. Construct a video activity tree T: $T := (Root, \varnothing, \{V_i \mid 1 \le i \le N\}, \varnothing)$, such that:
   $\forall\ V_i$: $Parent(V_i) = Root \wedge \forall\ i, j$: $1 \le i < j \le N \Leftrightarrow V_i <_t V_j$.

Next, an operation for editing video clips as a whole or for modifying their parts is presented:

Definition 8 (editing algorithm) The editing algorithm takes as input an identifier id of the clip and a range r of frames within it, which have to be replaced with a new video segment vs representing the results of performed modifications. The algorithm splits the affected node of the video activity tree T (if necessary) and accordingly updates its activity history:

1. Find a node $V \in T.Valid$: $V.NID = id \wedge r \subset V.Range$,
   Report failure if such node does not exist.
2. *If* $r = V.Range$, *Then* modify V:
   $V := (V.NID, [1, Length(vs.F)], vs, \{O \in V.History \mid O.Status = Shared\})$,
   Append an instance $(private, NewID(), Edit, \{id, r, vs\}, \varnothing)$ to $V.History$,
   Report success.
   Else construct an intermediate node $V' := (V.NID, V.Range, V.History)$,
   And go to step 3.
3. Construct valid nodes V_i (at least two of them with non-empty frame sequences):
   $V_1 = (V.NID, [V.Range.Low, r.Low - 1], LeftSegment, \varnothing)$,
   $V_2 = (NewID(), [1, Length(vs.F)], vs, \varnothing)$,
   $V_3 = (V.NID, [r.High + 1, V.Range.High], RightSegment, \varnothing)$,
   where the contents of video segments *LeftSegment* and *RightSegment* is illustrated in figure 2.
4. Adjust T: $T := (Root, T.Nodes \cup \{V'\}, T.Valid \cup \{V_i\} \setminus \{V\}, T.Dead)$, such that:
   $Parent(V') = Parent(V) \wedge \forall\ V_i$: $Parent(V_i) = V' \wedge \forall\ i, j$: $i < j \Leftrightarrow V_i <_t V_j$.
5. Append instance $(private, NewID(), Edit, \{id, r, vs\}, \varnothing)$ to $V_2.History$.

Evidently, the above algorithm provides an efficient means of modeling frame level manipulations within a particular video clip and at the same time supports operations applied to the whole segment, as already proposed in [11]. Since intra-segment editing is based on the node splitting technique, the affected valid node is replaced with a subtree, whose leaves are arranged to correspond to the resulting video material, as illustrated in figure 2. Moreover, propagation of the original node identifier to the node's respective sub-segments formed after splitting enables concurrent intra-segment manipulations in non-overlapping clip regions.

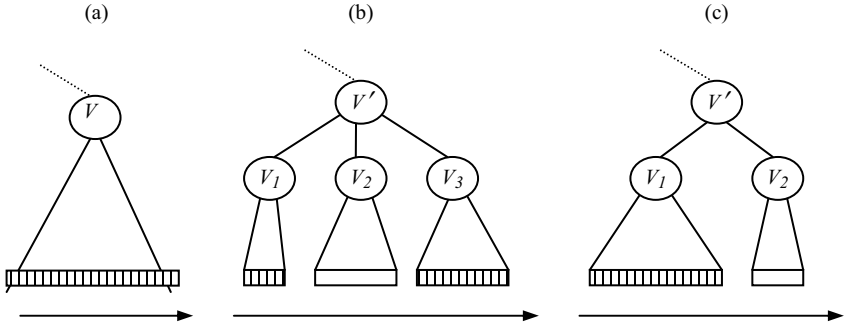


Figure 2. Node splitting carried out during intra-clip editing: (a) – an original valid node with the associated clip, (b) – split node after editing inside of the clip, (c) – split node after right-edge editing.

At this point it is worth to mention that editing algorithms make no modifications in the source video material – only references to the raw data are actually manipulated. Implicitly, such approach assumes the possibility of random access to the frames contained in the underlying video clips. However, this may be unfeasible (at least directly) when clips are coded with compression algorithms exploiting motion estimation, i.e. dependencies between adjacent frames. To overcome this difficulty it can be assumed that video material is accessed from an appropriate starting point, such as the nearest previous key frame, if necessary.

Next, operations for including new material into the video production and removing unnecessary parts from it are described. The insertion algorithm is left almost unchanged [11], while the deletion method now supports intra-clip removals.

Definition 9 (insertion algorithm) This algorithm takes as input a new video segment *vs* and its desired location on the timeline, specified by a destination point *pos* within a clip having given identifier *id*. The affected node is split and the activity history is accordingly updated:

1. Find a node $V \in T.Valid$:
   $V.NID = id \wedge (pos \in V.Range \vee (pos = 0 \wedge V.Range.Low = 1))$,
   Report failure if such node does not exist.
2. Construct valid nodes V_i (at least two of them with non-empty frame sequences):
   $V_1 = (V.NID, [V.Range.Low, pos], LeftSegment, \varnothing)$,
   $V_2 = (NewID(), [1, Length(vs.F)], vs, \varnothing)$,
   $V_3 = (V.NID, [pos + 1, V.Range.High], RightSegment, \varnothing)$,
   and split the node V in a way similar to splitting in the editing algorithm.
3. Append instance $(private, NewID(), Insert, \{vs, id, pos\}, \varnothing)$ to $V_2.History$.

Note, that in contrast to [11], insertion operation instance is stored in the newly created node and not in the split one. Along with propagating of the original segment identifier to its remainders, this enables commutativity of insertions occurred at different positions in the same clip.

Definition 10 (deletion algorithm) This algorithm takes as input an identifier *id* of the clip and a range *r* of frames within it, which need to be removed. The algorithm splits the affected node (if necessary), forms a new dead node and accordingly updates its activity history:

1. Find a node $V \in T.Valid$: $V.NID = id \wedge r \subset V.Range$,
   Report failure if such node does not exist;
   Coordinate with previous deletions and undo local insertions, like in [11].
2. Construct valid nodes V_i (at least two of them with non-empty frame sequences):
   $V_1 = (V.NID, [V.Range.Low, r.Low - 1], LeftSegment, \varnothing)$,
   $V_2 = (V.NID, r, \varnothing)$, this should be actually a dead node,
   $V_3 = (V.NID, [r.High + 1, V.Range.High], RightSegment, \varnothing)$,
   and split *V* in case its part is removed, otherwise *V* becomes a dead node itself.
3. Append an instance $(private, NewID(), Delete, \{id, r\}, \varnothing)$ to $V_2.History$.

Similar to [11], the deletion algorithm can be designed to support commutativity between any two removals, even overlapping within the same clip. Also, it can act as an inverse for some local insertions, which are not shared by other users and are not affected by subsequent operations. The latter can be seen as an example of history reduction at the user's side.

In addition to the presented editing operations it turns out to be useful to provide a dedicated moving algorithm, which would enable moving of selected contiguous blocks of the media stream to the specified locations. Though at first sight it may seem that such method is unnecessary because it looks the same as an appropriate sequence of deletion and insertion, in fact it is required since delete-insert pairs are in conflict with concurrent editing of the moved block. Moreover, moving and editing activities over a single segment can be regarded as compatible from a semantic point of view and thus the following algorithm has to be provided.

Definition 11 (moving algorithm) This algorithm takes as input an identifier *mid* of the clip and a range *r* of frames within it, which has to be moved. Their target location is specified by *pos* and *id* parameters, which are similar to their counterparts in the insertion algorithm. The algorithm is especially designed in a way to ensure compatibility of moving operation with concurrent manipulations within the moved segment, such as editing or insertion activities. To achieve this, it is insufficient to move a single valid node like in [11], but it is necessary to move a proper valid (if it exists) or the nearest intermediate node with a certain part of its subtree, which stands for concurrent manipulations mentioned above (if any). Throughout this process, the identifier of the moved clip should be kept intact, as outlined below:

1. Find a node $M \in T.Valid \cup T.Nodes$: $M.NID = mid \wedge r \subset M.Range$; or fail.
2. Apply steps 2, 3 of deletion algorithm to *M*, using *r* and *Move* operation instance.
3. Apply insertion algorithm with *M.Segment* (if *M* is valid node), *r*, *id*, and *pos* as input, such that $V_2 = (mid, r, M.Segment, \varnothing)$, and use *Move* operation instance.
4. Re-execute operation instances which were located in the nodes below *M* (if any);
   This step is required only when *M* turns out to be an intermediate node.

4. Achieving Cooperation

According to the principles of the CoAct transaction model, two distinct types of commutativity, namely forward and backward, are utilized for manipulating activity histories [9, 19].

In particular, backward commutativity is intended for determining dependencies between user's actions within a single history. The point is that the behavior of editing operations may be influenced by previously executed methods, for example, any modifications of a particular video clip depend on the preceding insertion of this clip into the media stream. Consequently, no operation instance can be exchanged between any workspaces without its relevant predecessors. And backward commutativity allows us to identify closed subhistories [9] having no external dependencies and thus representing consistent units of work, which can be exchanged between cooperating users separately from each other.

Actually, such dependencies are naturally reflected by the hierarchical structure of the original as well as the extended concurrent video model – dependent operation instances can be found "above" the given one within the tree. Hence, already developed algorithm for subhistory extraction [11], which is almost as simple as subtree selection, remains valid for the given proposal, providing an efficient way for identifying consistent units of work.

After extraction of consistent units of work from users' workspaces, another type of commutativity – forward – is exploited for detecting conflicts between operations coming from different users, serving as a basis for semantically correct merging of the results of individual activities. Forward commutativity properties for the methods of the data model presented in this paper are summarized below.

Operations	*Edit*(*ID*, *R*, *VS*)	*Insert*(*VS*, *ID*, *POS*)	*Delete*(*ID*, *R*)	*Move*(*MID*, *R*, *ID*, *POS*)
Edit(*id*, *r*, *vs*)	$id \neq ID \vee r \cap R = \varnothing$			
Insert(*vs*, *id*, *pos*)	$id \neq ID \vee pos \notin R$ *	$id \neq ID \vee pos \neq POS$		
Delete(*id*, *r*)	$id \neq ID \vee r \cap R = \varnothing$	$id \neq ID \vee POS \notin r$ *	*true*	
Move(*mid*, *r*, *id*, *pos*)	$(mid \neq ID \vee r \supset R \vee r \cap R = \varnothing) \wedge (id \neq ID \vee pos \notin R$ *)	$(mid \neq ID \vee (r.Low - 1 \neq POS \wedge r.High \neq POS)) \wedge (id \neq ID \vee pos \neq POS)$	$(mid \neq ID \vee r \cap R = \varnothing) \wedge (id \neq ID \vee pos \notin R$ *)	$\{mid, id\} \cap \{MID, ID\} = \varnothing$

* Actually, it is also reasonable to treat these operations as conflicting when $pos + 1 \in r$ in order to completely avoid commutativity in boundary cases

Table 1. Forward commutativity relations (symmetric).

Relations above demonstrate that editing operations of the expanded concurrent video model provide visible concurrency benefits comparing to the original proposal. It is now clear that users can freely work on the same segments until frame regions they are manipulating do not overlap and what is more, video extracts moved along the timeline can be simultaneously modified by other participants.

5. Conclusions and Further Work

In this paper, the concurrent video data model has been extended by means of the development of intra-segment editing operations.

Such extension has enabled a higher level of concurrency among cooperating users, at the same time providing more accurate support for modeling their activities in collaborative video authoring environments. Additionally, major transactional aspects of the original proposal [11] were preserved in a way to ensure efficient information sharing and exchange.

Moreover, the results of this work are general enough to be applicable to stream data in general. For instance, by renaming of frame sequences and video segments to sample sequences and audio segments, the concurrent video model can be transformed into concurrent audio, which may be found useful in the similar authoring environments.

As a possible direction of further research, investigation of versioning support for stream data on the basis of this work is considered.

Besides, development of a prototype system demonstrating feasibility of the presented approach is regarded as an essential part of the future work.

For this purpose, there is an intention to utilize the recently developed Advanced Authoring Format [7, 15] and its open source SDK. Basically, the AAF file format is an industry-driven standard especially designed for interchange of compositional meta-data between various multimedia authoring tools. At the moment of writing this paper, several video editing applications have already claimed a certain level of compliance with AAF. In particular, Adobe Premiere Pro [2] is able to export its projects in this novel form.

Acknowledgements

Special thanks are expressed to Boris Novikov for encouragement and discussions on the topic.

References

[1]. Adali, S., Candan, K.S., Chen, S., Erol, K., Subrahmanian, V.: The advanced video information system: Data structures and query processing. Multimedia Systems, Vol.4, pages 172-186, 1996

[2]. Adobe, Adobe Premiere Pro 7.0: http://www.adobe.com/products/premiere, 2004

[3]. Borghoff, U.M., Teege, G.: Structure management in the collaborative multimedia editing system IRIS. In Proc. of the International Conf. on Multi-Media Modeling, pages 159-173, Singapore, 1993

[4]. Bulterman, D.C.A., Hardman, L.: Multimedia authoring tools: State of the art and research challenges. In Lecture Notes in Computer Science #1000, Springer-Verlag, 1995

[5]. Davis, A.H., Sun, C., Lu, J.: Generalizing operational transformation to the standard general markup language. In Proc. of the 2002 ACM conference on Computer Supported Cooperative Work, pages 58-67, New Orleans, USA, November 2002

[6]. Dumas, M., Lozano, R., Fauvet, M.-C., Martin, H., Scholl, P.-C.: A sequence-based object-oriented model for video databases. Multimedia Tools and Applications, Vol.18, Issue 3, pages 249-277, 2002

[7]. Gilmer, B.: AAF – The Advanced Authoring Format. http://www.aafassociation.org/html/pr/291_gilmer1_0702.pdf

[8]. Ionescu, M., Marsic, I.: Tree-Based Concurrency Control in Distributed Groupware. Computer Supported Cooperative Work, Vol.12, No.3, pages 329-350, 2003

[9]. Klingemann, J., Tesch, T., Wasch, J.: Semantics-based transaction management for cooperative applications. In Proc. of the International Workshop on Advanced Transaction Models and Architectures, pages 234-252, Goa, India, August - September 1996

[10]. Mackay, W.E., Beaudouin-Lafon, M.: DIVA: exploratory data analysis with multimedia streams. Conf. proc. on Human factors in computing systems, pages 416-423, Los Angeles, USA, April 1998

[11]. Novikov, B., Proskurnin, O.: Towards collaborative video authoring. In Proc. of the 7th East-European Conference on Advances in Databases and Information Systems, pages 370-384, Dresden, Germany, September 2003

[12]. Oomoto, E., Tanaka, K.: OVID: Design and implementation of a video-object database system. IEEE Transactions on Knowledge and Data Engineering, Vol.5, No.4, August 1993

[13]. Rusinkiewicz, M., Klas, W., Tesch, T., Wasch, J., Muth, P.: Towards a cooperative transaction model - The cooperative activity model. In Proc. of the 21st International Conference on Very Large Databases, pages 194-205, Zurich, Switzerland, September 1995

[14]. Tesch, T., Wasch, J.: Transaction support for cooperative hypermedia document authoring – a study on requirements. In Proc. of the 8th ERCIM Database Research Group Workshop on Database Issues and Infrastructure in Cooperative Information Systems (EDRG-8), pages 31-42, Trondheim, Norway, August 1995

[15]. Turner, B.: A new take on teamwork. Video Systems, December 2002. http://videosystems.com/ar/video_new_teamwork

[16]. Ulead, MediaStudio Pro 7.0: http://www.ulead.com/msp, 2004

[17]. Wang, K.: The design of extending individual multimedia authoring to cooperative multimedia authoring. In Proc. of the Fourth International Conference for Young Computer Scientists (ICYCS'95), Beijing, China, July 1995

[18]. Wasch, J., Klas, W.: History merging as a mechanism for concurrency control in cooperative environments. In Proc. of RIDE-Interoperability of Nontraditional Database Systems, pages 76-85, New Orleans, USA, February 1996

[19]. Weihl, W.E.: Commutativity-based concurrency control for abstract data types. IEEE Transactions on Computers, Vol.37, No.12, pages 1488-1505, December 1988

[20]. Weikum, G.: Principles and realization strategies of multilevel transaction management. ACM Transactions on Database Systems, Vol.16, No.1, pages 132-180, March 1991

[21]. Weiss, R., Duda, A., Gifford, D.: Composition and search with a video algebra. IEEE Multimedia, Vol.2, No.1, pages 12-25, 1995

[22]. Xiao, B.: Collaborative multimedia authoring: scenarios and consistency maintenance. In Proc. of the Fourth International Workshop on Collaborative Editing Systems, New Orleans, USA, November 2002

SCIENTIFIC PAPERS UNIVERSITY OF LATVIA. 2004. VOLUME 672, Datbases and Information Systems
145–160 P.

The Middleware Support for Consistency in Distributed Mobile Applications*

Anna Kozlova, Dmitry Kochnev, Boris Novikov

University of Saint-Petersburg
Universitsky pr., 28, Peterhof, Saint-Petersburg, Russia
anet@ntc-it; dmitry@smartphonelabs.com; borisnov@acm.org

Abstract. This paper presents a model of the distributed middleware with transactional support. Our approach provides for high availability of the system in the fluctuated mobile environment and a high degree of a consistency when network connections are stable. The proposed set of the high-level operations allows high level of concurrency while processing XML-like data structures. A concept of the "accumulator" is introduced, providing for efficient conflict resolution during reconciliation.

Keywords: nomadic system, mobile transactions, high-level operations, accumulators.

1. Introduction

Business goes mobile. Cellular mobile networks allow for high-speed data transfer. Cellular networks coverage becomes more and more comprehensive. The number of high performance mobile devices which support complex applications is increasing from year to year. One of the most promising directions of the mobile IT-industry [1] is the development of applications with the vertical architecture known as business-to-employee (B2E) applications. Such applications provide solutions for interaction problems between the company and its employees, for example, enterprise resource planning (ERP), customer relationship management (CRM) or sales force automation (SFA).

The mobile environment imposes a number of restrictions on the applications:, such as limited sizes of memory and storehouse of the data, small computation power and small power resources, etc. [2].

The performance and communication characteristics of a cellular network may fluctuate depending on a place, time of day, weather, activity of subscribers and many other factors. A **stable cellular network** possesses good communication characteristics (i.e. "big enough" data amounts can be transferred "quickly enough" between the client and the server) and the intermittent connection (i.e. network connection can not be kept alive for a long time).

* This research was partially supported by RFBR grant. No. 04-01-00173

1.1. Problem statement

The basic problem which should be solved when creating a B2E-system is to adapt a well-known design of wired solution, which uses wired business processes for the mobile environment. The usual, albeit not the best, approach frequently used in the industrial applications, is as follows: the conflicts which arise after disconnection in such applications, are resolved by using one of the predetermined rules, for example, that the user should choose a solution for each conflict manually.

In this article we propose a solution, which involves both application-transparent and application-aware adaptation approaches [3].

The system, which is considered in the scope of this paper, has a nomadic architecture [4], and the clients do not communicate directly among themselves. The primary server is a high performance computer, which manages numerous connections and large amounts of data. The client manages a local replica of data and allows disconnected operation. Technological restrictions and features related to implementation of client application for such system are described in detail in [2].

The purpose of this paper is to design the prototype of middleware for the mobile distributed system with the support of transactions, which provides high level of availability of system in the unstable (fluctuated) mobile environment and high degree of a consistency of the data in the environment of a stable mobile network.

2. Related Works

Distributed systems include **traditional** distributed systems, **nomadic** distributed systems, and **ad-hoc** mobile distributed systems. The current document is focused on the concept of the middleware for nomadic systems. The detailed middleware review is resulted in [4].

2.1. Middleware

Most of existing middleware technologies, such as object-oriented middleware [5], message- and transaction-oriented systems, hide from the application the heterogeneity and the distributed nature of its environment. This approach, although very efficient and cost-effective in "wired" environment, is not well-suited for the systems that work in wireless environment [6].The basic problems arising at moving of the system to a mobile environment are synchronization of the data (i.e. restoration of data consistency) and maintenance of availability (inapplicability of data locking) [7].

One of the major issues targeted by data-sharing middleware systems such as Bayou [8], Coda [9], its successor Odyssey [10] and Xmiddle [11], is the support for disconnected operation and data-sharing.

The proposed solution inherited some features from Bayou [8], but the proposed solution supports the correct histories on the clients and on the server and allows processing the reconciliation and further replaying of histories instead of applying replica merging.

As Odyssey [10] the proposed solution introduces context-awareness and application-dependent behaviors. It focuses on efficiency of data granularity and improves Odyssey's architectural solution on application-aware approach by offering server-based accumulator abstraction.

Xmiddle [11] allows mobile hosts to share data when they are connected, or replicate the tree-like data and perform operations on them off-line, when they are disconnected. Reconciliation policies are specified as part of the XML Schema definition of the data structures that are handled by Xmiddle itself. The weakness of such system is that the policies of reconciliation are statically defined and strongly related to the data structures used by the application. We improve this approach in such a way that the application may contain additional server-side units implementing conflict resolution policies as accumulators.

Tuple space based systems for logical and physical mobility such as JavaSpaces [12], Lime [13], Limbo [14], and IBM T Spaces exploit the decoupling in time and space of these data structures in the mobility context. Tuple-spaces are multi-sets, which means that every tuple can be duplicated in the space. Tuple spaces are very general concept that looses data structures, so this approach cannot be applied to the highlighted problem because it has irresolvable disadvantages with data reconciliation.

2.2. Mobile Transactions

The classical concept of transaction is not applicable in a mobile environment [6, 7]. There are some approaches, which take into account the specifics of mobility, expanding classical definition. Most approaches use replication of the data on a client part of the application, which raises data availability. Pessimistic approaches [15] do not support replication and disconnected operation, and thus cannot be applied to the highlighted problem.

The model described in this paper uses the optimistic approach. The close approaches are presented in [16] and [17], where the commitment is processed on a local replica, and then the lifecycle of the application proceeds in the assumption that the commitment will be confirmed by the server. In [16] the results of the same transaction processing on the client and server sides may differ; and the correctness of the result defines by client application.

As well as the approaches using broadcasting [18, 19], our approach reduces ascending traffic (from the client to the server). For this purpose read-only transactions are being committed on the client side without any confirmation from the server. The local serialization graph is used for recognition of transactions that in any way cannot be serialized on the server. The similar ideas are represented in the approach of [20] where the server computes so called dependency information and broadcasts it to clients.

The detailed review focused on the computational model and ACID-properties of approaches to mobile transactions are represented in [21].

2.3. Data Model & Operations

Instead of Document Object Model (DOM) by World Wide Web Consortium, the offered data representation uses unique identifiers for each node due to distributed and replicated nature of the data.

There are a lot of data models, which have been known for a long time, that use the abstract high-level operations defined as sequences of atomic operations. The concept of multi-level transaction [22] is most interesting for our purpose. In most cases the specially defined high-level operations commutate with each other in spite of the elementary operations laying in their basis. Use of the similar approaches raises concurrency of transactional operation in the system.

The data considered in this paper may have a non-numerical nature and complex internal structure, therefore some semantic conflicts arising between operations can not be represented as pseudo-conflicts, and thus universal commutativity can not be achieved as in pseudo-conflicts between *Withdraw()* and *Deposit()* operations working with account [22]. The decided problem is to define the operations so that the number of irresolvable conflicts would be as little as possible.

In [23] the ideas of the concurrent video model related to high-level operations and unique identifiers applied to the linear data model. We admit that the further development of these ideas will allow applying the similar approach to the XML-structures.

3. Data Model

The data using by most B2E -applications (e.g. sales force automation) may be represented as tree-like structures semantically associated to graphs, which are stored in XML documents. Naturally, the organization consists of departments; employees work in departments; each department develop some projects, related to a client base; each projects relates to a group of employees; clients are related to the projects, etc. Tree structure allows sophisticated manipulations due to the different node levels, hierarchy among nodes, and the relationships among the different elements, which could be defined.

The data used in the internal representation turn out from the normal XML by applying the transformation, which moves the attributes into the affiliate nodes (Figure 1). Thus, the target XML has the following properties:

- Non-leaf nodes contain the navigational information only.
- The data values are stored only in the leaf-nodes.
- The references are stored only in the leaf-nodes. Each reference-node points to no more than one node. Also, the reference-nodes point to navigational nodes only.

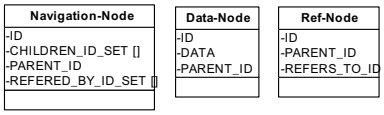


Figure 1. Node structure

Moreover, each node is assigned to the unique identifier due to distributed and replicated data representation. This unique identifier is kept in all replicas.

A replica presents determined by a subscription a subtree of the tree of the master replica (Figure 2.1, 2.2). If the subscription contains a pair of nodes, so that one node is an ancestor of another, then nodes of intermediate generations also will be included in a subscription.

By default, the data in leaves stores according to the concept of application-transparent adaptation [3]. In addition to this, the system allows to organize access to some data items according to the application-aware concept. For this purpose the leaf of a tree should detail the "accumulator" abstraction offered by us in Section 5. The client's replica contains only the elements of a simple type (e.g. a number or a string) instead of accumulators.

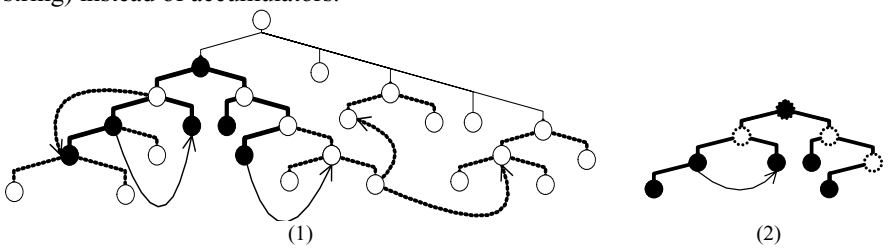


Figure 2. Client's replica as a part of server's replica (1) and the replica on the client side (2)

4. Operations

In the model described in this paper the following operation above tree structures are defined: *insert_node, insert_data,* and *insert_ref*, *delete*, *update*, *select* and *move*. The operations defined in such way are commutating with each other in most cases. The formal definitions are presented further.

Definition. The **insert operation** *(Insert(parent_id, sibling_id): newID)* operation inserts a new node in the data tree as a child of the node with *parent_id* identifier and as a next sibling of the node with *sibling_id* identifier. If *sibling_id* is not set, the operation creates the first child of the node with *parent_id* identifier. If any required node is not found or has inappropriate type, this operation reports failure.

Definition. The **insert operation for navigation nodes** *(InsertNode(parent_id, sibling_id): newID)* inserts a navigation node using *Insert()* operation.

Definition. The **insert operation for data nodes** *(InsertData(parent_id, sibling_id, data): newID)* inserts a data node using *Insert()* operation according to the following algorithm:

1. Execute *Insert (parent_id, sibling_id)*
2. Associate the created node to the *data* entry

Definition. The **insert operation for reference nodes** *(InsertRef (parent_id, sibling_id, toID): newID)* inserts a reference node using *Insert()* operation according to the following algorithm:

1. Execute *Insert (parent_id, sibling_id)*
2. Find a node with *toID* identifier. Report failure if search failed

3. Associate the created node to the node with identifier *toID*
4. Insert the identifier of created node into *REFERED_BY_ID_SET*

Definition. The **update operation for data-nodes** *(Update(id, new_data))* sets the data entry associated to the node with id identifier to *new_data*. If the required node is not found operation reports failure.

Definition. The **delete operation** *(Delete(id))* removes the full subtree with the root node identified as *id*.

1. Find a node with identifier id. Set it as current node
2. Remove current node from the *CHILDREN_ID_SET* of its parent node
3. If this node is a reference (*REFERS_TO_ID* is not empty) then removes current node from *REFERED_BY_ID_SET* of the node with *REFERS_TO_ID* identifier
4. Apply Delete() recursively for each reference nodes listed in *REFERED_BY_ID_SET* of the current node
5. Apply Delete() recursively for each node listed in *CHILDREN_ID_SET* of the current node
6. Remove the current node

Definition. The **select operation** *(Select(id))* returns the node with identifier id as an object or reports failure if the required node does not exist.

Definition. The **move operation** *(Move(id, new_parent_id, new_sibling_id))* sets the subtree with root node identified by as a child of the node with *new_parent_id* identifier and as a next sibling of the node with *new_sibling_id* identifier. If *new_sibling_id* is not set, the operation inserts the first child of the node with *new_parent_id* identifier. The provisional algorithm follows:

1. Find nodes with identifiers *id, new_parent_id, new_sibling_id.* Report failure if such nodes do not exist.
2. Assign a node with *new_parent_id* identifier as a parent to a node with identifier id
3. Insert *id* value after *new_sibling_id* into the *CHILDREN_ID_SET* of the node with *new_parent_id* identifier.
4. Remove identifier *id* from *CHILDREN_ID_SET* of the old parent of *id*

	InsertNode	InsertData	InsertRef	Update	Delete	Select	Move
InsertNode	T	T	T	A	T*	T	T
InsertData	T	T	T	T	T*	T	T
InsertRef	T	T	T	A	T*	(1)	T
Update	A	T	A	T	T*	T	A
Delete	T*	T*	T*	T*	A	(2)	T*
Select	T	T	(1)	T	(2)	A	T*
Move	T	T	T	A	T*	T*	A

Table 1. Commutativity table.

The Legend.

≠# Additional assumptions for the pairs of operations marked as T* are as follows:

$$Op_1(X) \text{ ⁄ } Op_2(args) == Op_2(args) \text{ ⁄ } Op_1(X) \text{ only if } X \supset Parent(args)$$

where *Parent(args)* is a conjunction of parent sets of all elements of *args*, and Op_1 is *Select()* or *Delete()*.

≠# Additional conditions dependent on the parameter values of the operations (and independent on database state) are required in the cases marked as (1) and (2)

4.1. Commutativity Table

The above table describes the commutativity aspects of proposed operations. The pairs of operations that always commutate marked in the table as *A* (*ALWAYS)*. If the disjunction of the argument sets of both operations is empty, the pairs of operations marked in the table as *T* (*TRUE)* will commutate. Naturally, the commutativity between some operation and *insert*()* requires a weaker assumption: the *NewID* generated by *insert*()* should not equal to any argument of the second operation.

Thus, the proposed high-level operations in most cases commutate with each other in spite of the elementary read and write operations laying in their basis. For example, operations *Move()* and *InsertNode()* may be represented as a composition of elementary read-write operations as follows:

```
Move(id, new_parent_id,
     new_sibling_id) °
     ° Read(id) Ŭ
     Ŭ Read(new_parent_id) Ŭ
     Ŭ Read(new_sibling_id) Ŭ
     Ŭ Write(id Ą getParentID()) Ŭ
     Ŭ Write(new_parent_id) Ŭ
     Ŭ Write(id);
```

```
InsertNode(parent_id, sibling_id):
     newID °
     ° Read(parent_id) Ŭ
     Ŭ Read(sibling_id) Ŭ
     Ŭ Write(parent_id) Ŭ
     Ŭ Write(newID);
```

If *id==parent_id* there is a conflict between elementary operations, instead of it, the proposed high-level operations commutate with each other.

5. Accumulators

Accumulators are intended for "intervention" of the application to the conflict resolution. As against simple types, the accumulator stores operations applied to some initial value of simple type instead of the explicit value of the element. Thus, accumulators allow on the air replaying the operations. It decreases number of conflicts between *update* operations, i.e. reduces number of transaction aborts in the system.

Definition. The **base value** is value of simple type. To compute the value of the accumulator on any moment of time one shall apply the operations stored in accumulator to that base value.

Definition. The accumulator stores operations in the **operations collection**. Each operation in the accumulator has two timestamps. The **insertion timestamp (ITS)**

defines the moment when this operation has been added into the accumulator; and the other one is the **execution timestamp (ETS)** that equals to the **client replica timestamp (CRTS)** that denotes a timestamp of last connection to the server with participation of the given client, because from server's point of view the client has no clock. The operations are defined and implemented at the application layer.

Definition. The **external functions** are intended to get additional information about the accumulator. External functions may depend not only on the operations and the base value of the accumulator, but also on other parameters, for example, another accumulator. An external function semantically **depends on the period of time**, if it depends on operations that have been applied during this period. The external functions are defined at the application layer.

Example. "The daily average balance on the account" may be an external function for the account represented as accumulator. The value of this function can be used for operation *Op* defined as "monthly interest of the account". The accumulator allows inserting into the history transactions that could not be inserted according to application-transparent approach. If after applying of the operation *Op* the server receives the operation Op_1 that influence to the result of applying *Op*, such operation will be just added into the accumulator. As a result, the current value provided by the accumulator, will allow for Op_1 applying, because this value is calculated "on the air".

5.1. Accumulator Design Details

Two typical designs of the collection are introduced: list- and set-accumulators. The application developer may also implement his own designs of the collection. The main difference between list- and set-accumulator collections is that in a list all elements are strictly ordered and in the set all elements are stored without any order.

Uniqueness of elements in set is not important in this context, because each operation received by the server has unique identifier. This identifier consists of unique identifier of the client that causes this operation and unique identifier of this operation on that client.

For each type of collection one of the proposed strategies of modification may be applied: the first strategy is so called insert-only strategy; the second one is insert-and-remove strategy. The methods of replaying the operations for both strategies are defined further. The application developer may choose between those methods according to time-dependence of external functions and conflict presence between defined operations.

5.2. Set-Accumulator

The time concept cannot be applied to sets, because it would define the artificial order. Also, the values of the external functions do not depend on the order of elements; it means that each operation in the set should commutate with each other.

Inserting or removing the new operations in such accumulator occurs as inserting or removing the element in the set. Replaying of some operation in set can be

implemented by two methods: as replacement of old operation with the new one, or as inserting a compensational operation and then inserting of the new operation.

Because of commutativity of operations, the results of each method are the same. The only difference is that in the first case the removed operation will not appear anymore in the accumulator; in the second case inserting the compensational operation requires its explicit existence.

5.3. List-Accumulator

Because of strictly defined order, each operation stored in the list-accumulator has a context of other operations. (By the way, the list-accumulator may be applied as a set, because each operation has its unique identifier, and the context may be ignored in the application.) There are two kinds of list-accumulators: insert-only and insert-and-remove list-accumulators. Inserting or removing the new operations in such accumulator occurs as inserting or removing the element in the set. The replaying strategies for the list-accumulator follow.

Definition. An **insert-only list-accumulator** supports only one way for replaying: insertion of a compensational operation and insertion of the new operation. There are following cases related to time-dependency of the external functions and the commutativity of the defined operations:

- **Case 1**. There are some time-dependent external functions in the accumulator
  - If *all operations in the accumulator commutate with each other*, inserting of the compensational and new operations occurs according to the rules of detection of the *ETS*.
  - Else, if *some operations in accumulator do not commutate with each other*, inserting of the compensational and new operations occurs according to the rules of detection of the *ETS*. Because the operations do not commutate, the application should warrant the semantic correctness of the replaying.
- **Case 2.** All external functions are time-independent
  - If *some operations in accumulator do not commutate with each other*, inserting of the new element in any place of the list does not damage the rules of detection of the *ETS* because all external functions are time independent. Thus, there are several cases to replay the operations:
    - Insertion of the compensational operation and the new operation just after the old operation
    - Insertion of the compensational operation just after the old operation and inserting the new operation according to its *ETS*
    - Insertion of the compensational and the new operations according to their *ETS*

The compensational operation and the method of replaying are defined by the application.

Definition. An **insert-and-remove list-accumulator** support two approaches for replaying: inserting of the compensational operation (the same way as insert-only case, considered above) and removing of the old operation. If some external function

depends on time some information related to the old operation may be lost, so this method damages the semantic correctness of such function.

- **Case 1.** All external functions are time-independent
  - If *some operations in accumulator do not commutate with each other*, inserting of the new element in any place of the list does not damage the rules of detection of the *ETS* because all external functions are time independent. Thus, there are several cases to replay the operations:
    - Removal of the old operation and inserting of the new one with old *ETS*
    - Removal of the old operation and inserting of the new one with the new (actual) *ETS*

The application defines the replaying method.

5.4. Purging of Out-of-Date Operations

While the system works the number of the operations in the accumulators grows. To improve the performance of the system, it is necessary to purge out-of-date operations and to replace the base value with the new one from time to time. The new value is a result of applying the operations that will be purged to the old value, but generally

$$F(b, (op_1, op_2, op_3)) \neq F(op_1(b), (op_2, op_3)) \quad (*)$$

This issue may be solved using additional assumption about the external function: it must depend on limited (explicit) period of time, i.e. depend on final number of operations. This period of time refers to the dependent window of the external function. Of course, this number may be great. Thus, the operation became out-of-date if its *ETS t* satisfies to the following predicate:

$$t \{ t_{current} \geq \max_{i \in 1..m} \{ t_i \mid t_i \geq \text{dependent window of } F_i \}$$

5.5. Accumulators Advantages

The offered strategies allow the application to define the criteria of a correctness using definition of the operations and external functions. Thus, the application is permitted to influence resolutions of conflicts.

Purging of the accumulator from old operations also may be initiated by the application, it allows the server to manage the instance of accumulator in the efficiently manner.

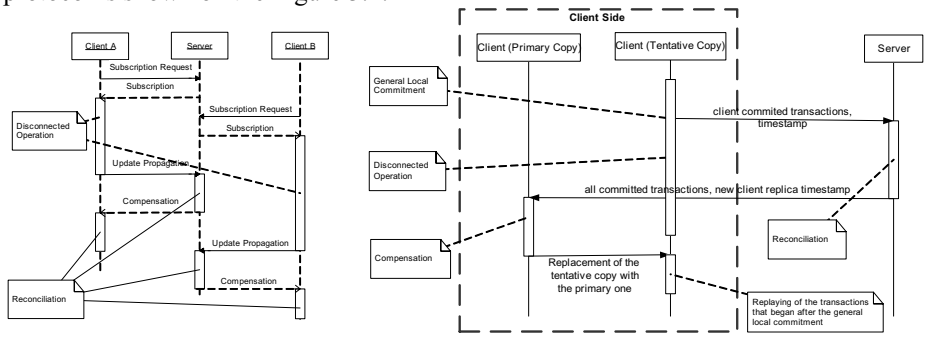
6. Protocol for mobile transactions

The proposed protocol is based on low-level protocol that supply data transfer between mobile hosts, for example, HTTP over GPRS or UMTS. The protocol must satisfy to the following conceptual requirements:

1. The server must reconcile histories from all clients and keep the master replica in consistent state.

2. The clients must propagate their changes to the server and receive from the server the changes made by the other clients that modify the local replica of this client.
3. To achieve the appropriate level of availability the client should operate in disconnected mode.
4. During disconnected operation the client should manage two local replicas of the data. The **primary replica** one should be identical to the master replica stored on the server at the moment of the last connection to the server. All changes during disconnected operation should be applied to the **tentative replica.**
5. The **local conflicts** (the conflicts on the tentative replica) must be resolved by the client without any participation from the server.
6. The client must commit read-only transactions.
7. To save its computational resources, the client should process the garbage collection, i.e. to clean the memory from the unused structures for conflicts resolving on tentative copy.

The typical scenario of interaction between hosts (Figure 3.1) gives a large scale view of the system work. First, the client subscribes for the data and receives from the server a replica marked with a special *CRTS*. When disconnection occurs, the client continue working with its replica Further, at some moment depending on the adaptation policy of the application, the client restores the connection and send its changes to the server. The server reconciles the information received from the client with the master replica and returns all changes that modify client's replica. Then client reconcile the received changes to its local replica. The large scale of the protocol is shown on the Figure 3.2.



(1) (2)

Figure 3. System lifecycle (1) and the protocol (2)

6.1. Client Side Algorithm

1. Disconnected operation
   - ⋢# Permanent concurrent execution of the transactions on the tentative copy and manage the local serialization graph
   - ⋢# If the server response that was requested in the previous execution of the step #2 has been received, go to step #3

- ∉# After returning form the step #3 and #4 *may* continue with the step #2

2. Updates propagation
   - ∉# Pause disconnected operations (Step #1)
   - ∉# General local commitment processing, i.e. commitment of all local transactions
   - ∉# Purge all update transaction from the serialization graph
   - ∉# Mark all read-only transactions
   - ∉# Send the *commit_request* that includes all update transactions and the CRTS to the server
   - ∉# Continue with the Step #1
3. Compensation
   - ∉# Receive the part global history that modifies the client's replica and a list of client's transactions aborted by server
   - ∉# Cascadely abort of all transactions that read from transactions that have been aborted by the server
   - ∉# Add the received transactions to the local serialization graph
   - ∉# Cascadely abort of all transactions that conflicts with transactions that have been committed globally
4. Reconciliation
   - ∉# Replay the history received from the server on the primary copy
   - ∉# Replace the tentative copy with the primary copy
   - ∉# Pause disconnected operations (Step #1)
   - ∉# Purge the serialization graph from all transactions with server timestamps and from read only transactions marked on the Step #2
   - ∉# Replay the transactions from the serialization graph on the tentative copy
   - ∉# Continue the disconnected operations (Step #1)

6.2. Server Side Algorithm

1. Receive the transactions from the client in serial order with the CRTS.
2. Insert the received transactions into the global history using histories merging procedure.
3. Send to the client all transactions that modify the local replica of that client and have not been sent to him yet. Send the list of local transactions that have been aborted by the server.

6.3. Histories merging procedure

The following algorithm is intended to the operation with non-accumulator arguments. Otherwise the accumulators' technique should be applied.

The operation *op* having timestamp *t* can be inserted into the history *H* if

$$\&op(i,x_i,t_i)\subset H,t_i\}\,t(op(i,x_i,t_i)\,\mathbb{A}\,op\,\sum op\,\mathbb{A}\,op(i,x_i,t_i)).$$

If it is true for each operation in the transaction then this transaction can be serialized.

The policy the transaction aborting may be defined on the application side. For example, it is possible to abort a transaction with the minimal timestamp. This

algorithm may be improved using the method [24], where for each data element the timestamp of last read, write etc. are stored in the special structures.

6.4. Approach to Information Selection for Transferring to the Client

To determine which transactions should be sent to the client, the server uses the *CRTS*. If the pre-image of the client's replica at the moment of the *CRTS* on the server does not contain any elements of accumulator type, the *CRTS* uniquely determines the data of this pre-image at the moment defined by the *CRTS*. Thus the server should send to the client all operations from the global history, which has a timestamp greater than the *CRTS* and that has the arguments from the pre-image of the client's replica at the moment of the *CRTS*.

The previous approach cannot be applied if the pre-image of the client's replica at the moment of the *CRTS* on the server contains some elements of accumulator type. For example, after the moment of the client A *CRTS* the client B inserts an operation with the *ETS* less than the client A *CRTS*, and this operation does not commutate with the subsequent operations. So, the value of the accumulator at the moment defined by the client A *CRTS* was changed, but it cannot be propagated to the client using the previous approach.

In the case of elements of accumulator type, the following approach is applied for reconciliation client's replica with the server's data. The server sends to the client not only the operations but also the values of the element having accumulator type at the moment of the *CRTS* only if there is an operation stored in the accumulator that is satisfying to the following predicate:

$$CRTS > ETS \;\&\&\; CRTS < ITS$$

This value should be included into the client's primary copy. Use of this approach warrants the appropriate level of consistency of the client and the server replicas.

To save the computational resources of the client host it is possible to transfer the whole new client's replica together with the operations. However, it brings an additional load onto the network.

6.5. Correctness of the protocol

Lemma 1. Inserting the read-only transactions into a global history does not influence its serializability:

$$\begin{aligned} &\& H_1 \; 4 \; \textit{the serial history of } C_1 \\ &\& H_2 \; 4 \; \textit{the serial history of } C_2 \\ &) \; H \; 4 \; \textit{the serial history} : H \neq H_1 \langle \; H \neq H_2 \end{aligned}$$

Proof.

1. Note that H_1 and H_2 differ only with the read-only transactions.
2. According to the proposed protocol the history
   $S_1 := (Op(H_1) \setminus \{op \subset Op(H_1) \mid op \subset$ *some read-only transaction of* $H_1\}, <_{H1})$
   is equivalent to
   $S_2 := (Op(H_2) \setminus \{op \subset Op(H_2) \mid op \subset$ *some read-only transaction of* $H_2\}, <_{H2})$
   (as it defined in [24]) and these histories are equivalent to some serial history according to the server side algorithm.

3. Insert all read-only transactions of H_1 and H_2 to S_1 (and denote it as S_1') and to S_2 (and denote it as S_2') according to its orders in H_1 and H_2
4. Since read operations do not conflict with each other, S_1' and S_2' are equivalent.
5. So, the global history S_1' containing all transactions of all clients has been found. Also, this history is equivalent to some serial history.

Lemma 2. The purging of the serialization graph on the client side is correct, i.e. the transactions, which would begin after the purging, would not conflict with transactions that have been removed from the graph.

Proof. On the client side the new transactions that began after the general local commitment has been processed are succeeding the transaction that has been purged from the local serializing after the commitment, so that any transaction will not go through local commitment. Thus, the purged transactions do not influence the processing of new transactions during disconnected operation.

Further, on the server side new transactions will receive the timestamp of the moment of the global commitment of the transactions purged on the client side before new transactions started. Thus, from the server point of view the new transactions are also succeeding the transactions that were locally committed on the client before.

Statement. Prove that the **assumption of general local commitment** of all transactions on the client side is essential. It means that it is impossible to warrant that purging of the serialization graph would not damage its integrity.

Consider the following example: the edge $t_1 \to t_2$ of the serialization graph was assigned to of the conflict between transactions t_1 and t_2 cause by operations $q_1(x)$ and $p_2(x)$. The transaction t_2 has been committed and purged from the serialization graph, but the transaction t_1 continue the executing. Then t_1 executes operation $q'_1(y)$ which has to cause the conflict with the operation $p'_2(y)$ of transaction t_2, but the transaction t_2 has been purged. Thus, the graph has to contain a cycle but it is an acyclic naturally.

Theorem. The proposed protocol is correct.

Proof.

1. The correctness of the server side algorithm is demonstrated by the procedure of merging histories at the server.
2. *Lemma 1* and *Lemma 2* prove the correctness of the client side algorithm.

7. Conclusion

The model of a middleware-system offered in this paper provides a model, which supports a limited form of consistency in a distributed mobile environment, where classical transaction models are not suitable. This approach is suitable for typical nomadic network and may be applied to a wide class of enterprise applications working in mobile environment.

Use of special set of high-level operations is one of the most important features of the introduced solution. These operations work with the data having a tree structure with XML-styled references. The key strength of the given model, in contrast with other approaches, is that changes of the replica transfer as high-level operations and not as simple data items. In most cases, two operations permute with each other. The

commutativity of those operations is based on their semantics that is defining by the target application. Thus, it reduces number of transaction aborts and, therefore,, availability and performance of the system increase in comparison with classical transactional operation in mobile environment.

Apparently, within the framework of the offered model, two update operations being the most often operation cannot be permuted. In general, a conflict like that may be resolved only on the application layer according to its semantics. We introduce abstraction of accumulator for effective implementation of the application-aware approach in the application layer.

According to this approach, the proposed model allows application to define the policy of adaptation for each node of the data tree. Unlike simple types, an accumulator does not store an explicit value, but it contains only the base value of a simple type and the collection of operations to be applied to the base value. The operations are determined on application layer as well as the external functions that return additional information about the node of accumulator type. In our prototype system there are two different implementations of accumulator concept called list and set. According to its semantics, the application may choose the suitable implementation for each node of accumulator type.

The distinctive feature of our system is the ability of client to resolve local conflicts without communication with server. To achieve the integrity of the client's replica, its representation includes external nodes with two-way navigation. These nodes help to recognize hierarchical relationship and allow the client to resolve local conflicts. Resolving of local conflicts improves the performance of client part of application and decreases communication cost.

The protocol we use has a significant limit: it requires general local commitment of all transactions before sending of any data to server. So, short local transactions should wait completion of long ones. That issue reduces efficiency of a client.

Our future research is focused on the issues related to replica and cache management, system scalability and system performance analysis. Also we plan to develop the strict theory of accumulators.

Acknowledgements

We thank Katja Perminova from SmartPhoneLabs, LLC for the comments which helped improve this article.

References

[1] D. Kochnev Features of mobile applications development, Mobile Communications/Russian Edition, No. 10, 2002 (in Russian).

[2] D.S. Kochnev, A.A. Terekhov Surviving Java for Mobile, IEEE Pervasive Computing, Vol. 2, No. 2, Apr-Jun 2003. P. 90-95

[3] M. Satyanarayanan, Fundamental Challenges in Mobile Computing, 15th ACM Symposium on Principles of Distributed Computing'96, Philadelphia, PA, USA, pp. 1-7, May 1996

[4] Cecilia Mascolo, Licia Capra and Wolfgang Emmerich. Mobile Computing Middleware, http://citeseer.nj.nec.com/596660.html

[5] W. Emmerich Engineering Distribued Systems, John Wiley & Sons, 2000

[6] L. Capra, W. Emmerich, and C. Mascolo. Middleware for Mobile Computing: Awareness vs. Transparency (position paper). In Int. 8th Workshop on Hot Topics in Operating Systems, May 2001.
[7] T. Imielinski and B. R.Badrinath, Mobile wireless computing: Challenges in data management, Communications of the ACM,Vol. 37, No. 10, October 1994, pp. 19--28.
[8] A. Demers, K. Petersen, M. Spreitzer, D. Terry, M.M. Theimer, and B. Welch. The bayou architecture: Support for data sharing among mobile users. In Proceedings Workshop on Mobile Computing Systems and Applications. IEEE, December 1994.
[9] Satyanarayanan, M, Kistler, J. J., Kumar, et. al., Coda: a Highly available File System for a Distributed Workstation Environment, IEEE Trans. on Computers, 39(4): 447-459, 1990.
[10] M. Satyanarayanan, Mobile information access, IEEE Personal Communications, vol.3, no.1, pp. 2633, 1996.
[11] C. Mascolo, L. Capra, S. Zachariadis, and W. Emmerich. XMIDDLE: A Data-Sharing Middleware for Mobile Computing. Int. Journal on Personal and Wireless Communications, April 2002.
[12] E. Freeman, S. Hupfer, and K. Arnold. JavaSpaces, Principles, Patterns, and Practice. Addison Wesley, 1999.
[13] Amy L. Murphy, Gian Pietro Picco, and Gruia-Catalin Roman. Lime: A Middleware for Physical and Logical Mobility. In Proceedings of the 21 st International Conference on Distributed Computing Systems (ICDCS-21), May 2001.
[14] N. Davies, S. P. Wade, A. Friday and G. S. Blair, Limbo: A Tuple Space Based Platform for Adaptive Mobile Application, Proceedings of the International Conference on Open Distributed Processing/Distributed Platforms (ICODP/ICDP '97), Toronto, Canada, 27-30 May 1997, pp291-302.
[15] S. Madria and B. Bhargava. A Transaction Model for Improving Data Availability in Mobile Computing, in Distributed and Parallel Databases, 10(2), 2001.
[16] J. N. Gray, P. Helland, P. O'Neil and D. Shasha. The Dangers of Replication and a Solution. In Conference on Management of Data, pp. 173-182, Canada, June 1996.
[17] E. Pitoura and B. Bhargava. Data Consistency in Intermittently Connected Distributed Systems. In Transactions on Knowledge and Data Engineeging, Nov. 1999.
[18] E. Pitoura and P. Chrysanthis, Scalable Processing of Read-Only Transactions in Broadcast Push, IEEE International Conference on Distributed Computing Systems, Austin, 1999
[19] J. Shanmugasundaram, et. al. Efficient Concurrency Control for Broadcast Environments Univ. of Massachusetts Technical Report 1999.
[20] Il Young Chung, Bharat Bhargava, Malika Mahoui, Leszek Lilien Autonomous Transaction Processing Using Data Dependency in Mobile Environments, In: Proc. of The Ninth IEEE Workshop on Future Trends of Distributed Computing Systems (FTDCS'03) p. 138, San Juan, Puerto Rico, 2003.
[21] P. Serrano-Alvarado, C. L. Roncancio et M. Adiba "Mobile Transaction Supports for DBMS" In: Proc. of 17ièmes Journées Bases de Données Avancées (BDA'2001) , Agadir, Maroc, 2001.
[22] G. Weikum, H.-J. Schek, Concepts and Applications of Multilevel Transactions and Open Nested Transactions, In Database Transaction Models for Advanced Applications, ed. A.K. Elmagarmid, Morgan Kaufmann, 1992.
[23] B. Novikov and O. Proskurnin. Towards collaborative video authoring. Proc. of the ADBIS'2003, 370-384, Dresden, Germany, 2003.
[24] P. Bernstein, V. Hadzilocs and N. Goodman, Concurrency Control and Recovery in Database Systems, Addison-Wesley, Reading, Mass., 1987.

XML AND DATABASES

SCIENTIFIC PAPERS UNIVERSITY OF LATVIA. 2004. VOLUME 672, Datbases and Information Systems
163–176 P.

A Matrix Model for XML Data[1]

Jaroslav Pokorny, Vladimir Rejlek

Charles University, Faculty of Mathematics and Physics
Malostranske nam. 25, Praha, Czech Republic
pokorny@ksi.ms.mff.cuni.cz

Abstract. Many modern applications produce and process XML data, which is queried in its both structural and textual component. To ensure such functionality, it is required to approach this data with both database and information retrieval methods. We start with a convenient vector space model and extend it with information about structural properties of an XML data collection *C*. According to the occurrences of a term *t* in the XML structure of *C* we represent *t* by a vector of weights. To reduce its lengths, a form of a DataGuide for *C* is considered with a path as a structure unit. Then, an XML document *D* is represented by a matrix ***D*** of weights. In the paper, we discuss possibilities of ranking in this model and adjusting ***D*** to reflect associations among paths in the DataGuide for *C*. Experimental results for a standard test collection are given and discussed.

Keywords. XML, vector space model, matrix model, ranking

1. Introduction

Many modern applications produce and process XML [15] data, which is queried in its both structural and textual component. This is especially useful if we consider a casual user who looks for information in web-based database systems or intranets containing XML data, like online shops, airline reservations, digital libraries catalogues or any other, and does not expect an exact answer. It is anticipated even that in the future, many websites will be built from XML documents. A remarkable characteristic of such XML collections is that they are mostly heterogeneous, e.g. they contain data from on domain but, possibly, valid w.r.t different DTDs or XML schemes. A part of these collections will be managed as XML databases [4] and will also provide a way for users to search their contents. To ensure such functionality, it is required to approach these websites with both database and information retrieval (IR) methods.

It is clear that IR approaches will be useful especially in cases of text-rich XML documents. As it is observed recently [9], current XML query languages like Xpath and XQuery are rather data-centric than document-centric. In other words, they are no longer appropriate for searching in such environments as they can not cope with the diversity of data. Hence, a research of integration of database querying and IR in context of XML is undoubtedly interesting and promising trend. Despite of the fact

[1] This research was supported in part by GACR grant 201/03/0912

that a variety of systems that support such methods have been proposed, conventional IR techniques [14] cannot be employed directly. The reason for it is that two types of queries should be dealt with: content-only queries, i.e. the traditional ones in IR, and content-and-structure queries.

Particularly, a number of techniques to extend the vector space model have been designed, e.g. [1,6,7,9,10]. In [9] so-called *indexing nodes* are explicitly indicated. They are disjoint and group some XML tree nodes. These structures are the basic units of indexing. The approach [6] uses indexing the pairs (*t*, *c*) through weights, where the term *t* is qualified by the context *c* in which it appears. The *c* depends on the path to *t.* Measuring of a context resemblance introduces another coefficients into query processing. In [10], a static index is built for basic indexing nodes and then, during query processing, vector space statistics are generated. A common property of these approaches is that these statistics are used to compute a similarity value for an XML document and a query and a subsequent ranking of these values.

Authors of [7] have proposed to use an extension of the traditional vector space model given by Fox [8]. Fox developed a method for representing in a single, extended vector different classes of information about a document, such as author name, terms etc. In [5] a weighting is applied to document fragment sequences that are manipulated by XQuery queries. The associated weighting algorithms are then fundamental for a rank operator.

A usual critique of the mentioned approaches is that they not sufficiently reflect the structure of XML documents. A more advanced, two-phase evaluation schema is proposed in [1]. First, a modified vector space model is employed to obtain similarity scores for the textual nodes of XML trees. Then, the scores are propagated upward in the XML-trees with a possible modification and possibly new scores of other nodes are generated.

We use a vector space ranking technique in context of the entire collection *C* of XML data. A document *D* is represented by a matrix $\boldsymbol{D}$, whose each row vector w^t associated with a term *t* contains the weights of *t* for each path occurring in *C*. For the path representation *C* we use a modification of well-known DataGuide technique [11]. Obviously, some weights in w^t will be equal to 0 dependent on whether *t* is contained in path or not. The query *Q* is also associated with a matrix $\boldsymbol{Q}$. The matrix model proposes to evaluate the degree of similarity of *D* with regard to the *Q* as the correlation between the matrices $\boldsymbol{D}$ and $\boldsymbol{Q}$. Experiments have shown that it is not possible to rely only on this score. Instead we adjust the matrix $\boldsymbol{D}$ by an additional data structure, so called a path transform matrix, which reflects relationships among paths in the DataGuide for *C*. Then, the resulted transformed matrices $\boldsymbol{TD}$ and $\boldsymbol{TQ}$ are used for query processing.

The rest of the paper is organized as follows. Section 2 repeats some basics about the vector space model. Section 3 describes its extension to the matrix model. A tree representation of the entire collection of XML data is defined here and associated weighting and ranking system are proposed. The notion of a path transform matrix is introduced and discussed in this section. In Section 4, we discuss some issues connected with the matrix model. In Section 5, we report our experiences in using the matrix model in the application domain given by the well-known Shakespeareôs collection. Finally, we conclude in Section 6.

2. Vector space model

Suppose a collection C of documents. In the vector space model, documents are treated as points in a high-dimensional space. As document $D_i \subset C$ is represented by a vector of term weights, i.e.

$$\boldsymbol{D}_i = [w_{i,1}, w_{i,2}, \acute{e}\ , w_{i,m}] \subset <0,1>^m$$

where m is the number of terms considered in C and i is the document number. The weight $w_{i,j}$ is usually calculated as $TF_{i,j}$ 1 $\log(N/n_i)$, where TF_{ij} is the number of times the term t_j occurs in a document D_i, n_i is the number of documents in which t_j occurs, and N is the total number of documents in the C. Similarly, a query Q can be expressed as

$$\boldsymbol{Q} = [q_1, q_1, \acute{e}\ , q_m] \subset <0,1>^m$$

To compare a query with a document, the most commonly used measure is the so-called *cosine similarity*, i.e., the cosine value of the angle between both vectors. So, we can define the similarity between a document D_i and a query Q as

$$Sim(D_i, Q) = \frac{\sum_{j=1}^{m} w_{i,j} * q_j}{\sqrt{\sum_{j=1}^{m} (w_{i,j})^2 * \sum_{j=1}^{m} (q_j)^2}}$$

3. Matrix model

We will now consider a collection C of XML documents containing large portions of text. Our goal is to model such data not only as XML trees, as usually, but also with respect to their text characteristics as it is usual in IR. Clearly, a pure vector model is not enough, as it does not reflect a structure of XML data.

We will use a lightweighted tree-like data model for handling XML documents, in which we create nodes for elements and attributes. Edges model the containment relationship between elements. The textual content of a element E with subelements (so called mixed content) is represented as a set of child nodes of the E node. Leafs nodes in the tree contain a piece of text (possibly empty). We neglect a difference between elements and attributes. In other words, we deal with attributes as with elements in the model.

Our matrix model is based on the vector space model. It only changes the meaning of the term weight in a document. In the vector space model a weight is expressed by a real number specifying a term weight for the entire document. In the matrix model, the term weight is expressed by a vector. Such a weight should reflect a distribution of the term in the XML structure of the document. Then a document D_i is represented as a matrix

$$\boldsymbol{D}_i \mid \begin{pmatrix} w_{i,1,1} & w_{i,1,2} & \dots & w_{i,1,k} \\ w_{i,2,1} & w_{i,2,2} & \dots & w_{i,2,k} \\ \dots & \dots & \dots & \dots \\ w_{i,m,1} & w_{i,m,2} & \dots & w_{i,m,k} \end{pmatrix} \subset \{ 0,1 \}^{m,k}$$

where k denotes the number of paths in the XML structure of the entire document collection C. The value $w_{i,j,s} \subset <0,1>$ is the weight of the term t_j on the path s in the document D_i.

3.1. DataGuide for collection

To apply the above ideas for a collection C of XML data, we need a data model of C. A tree representation can be used also in this case. Such tree can be modelled by the well-known DataGuide [11]. The DataGuide has been developed for the OEM data model originally [2]. Transferring the notion of DataGuide to the XML data model is in principle straightforward. One must however decide whether the DataGuide describes only the primary structure or the secondary structure as well, i.e., constructs such as IDREF. We easily will not assume mutual references among XML documents in our approach.

The idea of a DataGuide is to provide a summary of the structure, generated from the collection C. A *DataGuide for a collection C*, DG_C, of XML documents is a tree T_C fulfilling the following conditions:

- each path in C exists in T_C,
- each path in T_C exists in C,
- paths in T_C are unique (but not in C).

Notice, that in the case of non-existence of a common root it is possible to add formally such node to the forest of trees and get the required tree. Every leaf node of the tree is *annotated* with the IDs of those document leafs it represents (i.e. those reached by the same label path as the index node). In fact, a leaf of the tree is associated with a bag of element contents (PCDATA). Remind that DG_C is not given uniquely in general and does not support ordering in the XML documents in C.

Example 1: Consider a collection C_1 with three documents in Fig. 1. We index terms "David", "Morrell", "Caine". The associated DataGuide DG_{C1} and the list of its paths are depicted in Fig. 2.

©

The resulting tree is usually much smaller than the forest of document trees constructed from C (although theoretically its size is linear in that of the document forest). Hence it can be supposed to be held in main memory even for large document collections.

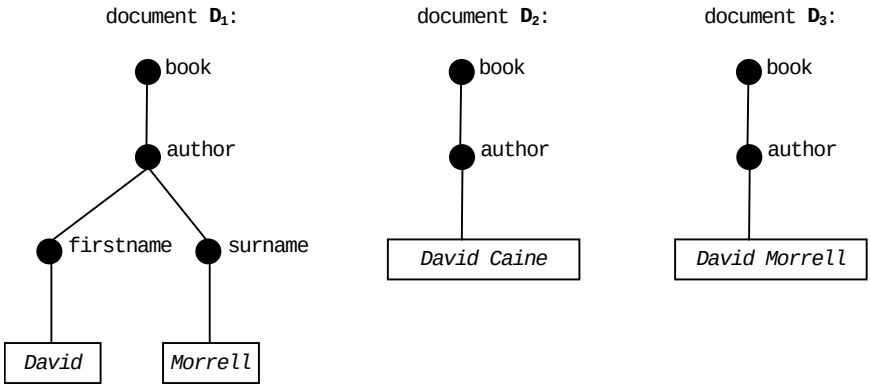


Figure 1. Collection C_1 of XML documents.

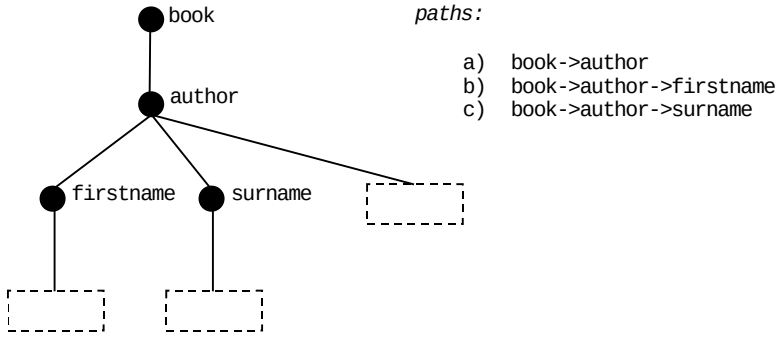


Figure 2. DataGuide for C_1.

3.2. Weighting and Ranking

Assuming a DG_C we now try to construct a matrix. Given a term t, path p, and document D_i, the term weight $w_{i,t,p}$ is defined as

$$TF_{t,p} / m_p$$

where $TF_{t,p}$ is the number of t occurrences on p and m_p is the number of all terms associated with p. Our matrix model will be used for tree queries, which cover the core XPath [16] constructs. Consequently, we express such queries in a similar way as documents, i.e. as matrices. For an evaluation of queries we will inspire by the original vector space model and introduce a similarity Sim_1 as

$$Sim_1(D_i, Q) = \sum_{l=1}^{m} \sum_{j=1}^{k} d_{i,l,j} * q_{l,j} \qquad (1)$$

where i, m, and k have the same meaning as above.

Example 1 (continued): The (3-dimensional) matrix ***D*** for collection C_1 can be represented in the following form.

	a	b	c	a	b	c	a	b	c
D_1:	[(0	**1**	0)	(0	0	**1**)	(0	0	0)]
D_2:	[(**0,5**	0	0)	(0	0	0)	(**0,5**	0	0)]
D_3:	[(**0,5**	0	0)	(**0,5**	0	0)	(0	0	0)]
	"david"			"morrell"			"caine"		

We evaluate the queries in Fig. 3 in this model.

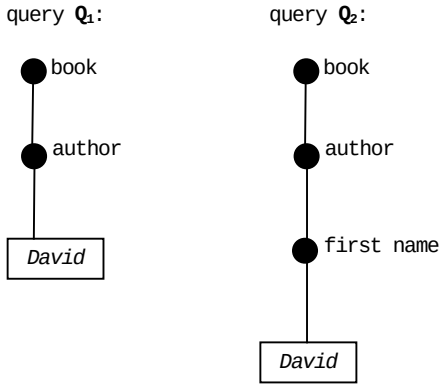


Figure 3. Queries Q_1 and Q_2

Thus, we obtain matrices

	a	b	c	a	b	c	a	b	c
Q_1:	[(1	0	0)	(0	0	0)	(0	0	0)]
Q_2:	[(0	1	0)	(0	0	0)	(0	0	0)]
	"david"			"morrell"			"caine"		

Then, we get the results

$\mathrm{Sim}_1(D_1, Q_1) = \mathbf{0}$
$\mathrm{Sim}_1(D_2, Q_1) = \mathbf{0.5}$
$\mathrm{Sim}_1(D_3, Q_1) = \mathbf{0.5}$
$\mathrm{Sim}_1(D_1, Q_2) = \mathbf{1}$
$\mathrm{Sim}_1(D_2, Q_2) = \mathbf{0}$
$\mathrm{Sim}_1(D_3, Q_2) = \mathbf{0}$

©

The results based on the Sim_1 are not too successful in Example 1. Particularly, $Sim_1(D_1, Q_1) = Sim_1(D_2, Q_2) = Sim_1(D_3, Q_2) = 0$ is not in accordance with our intuition. The reason is that for different XML structures their similarity is equal to

0. Since our goal is to process documents from heterogeneous sources, e.g. with different DTDs, it is necessary to add another additional structure to our considerations which will enable to describe associations (similarity) among paths in DG_C.

3.3. Path Transform Matrix

By a path transform matrix we understand a square matrix ***A*** of dimension $k \Delta k$, where k is the number of paths in DG_C, $a_{i,j} \subset <0,1>$, and $a_{i,i} = 1$. For each path the matrix **A** contains a vector that expresses a relationship (similarity) of the path to the other ones. This similarity is expressed by a real number from <0,1>, where 0 means that paths have nothing common and 1 means that the paths are the same.

Example 1 (continued): The path transform matrix could have a form

A_1	a	b	c
a	1	0	0
b	0,5	1	0
c	0,5	0	1

We can observe, that the weights on the path `book⇓ author⇓ firstname` have to be 0.5 times transformed on the path `book⇓ author`. Similarly weights on the path `book⇓ author⇓ surname` are transformed 0.5 times on the path `book⇓ author`. Recalculating all weights (see (2)) we obtain matrices ***TD*** and ***TQ***

	a	b	c	a	b	c	a	b	c
TD_1:	[(0,5	1	0)	(0,5	0	1)	(0	0	0)]
TD_2:	[(0,5	0	0)	(0	0	0)	(0,5	0	0)]
TD_3:	[(0,5	0	0)	(0,5	0	0)	(0	0	0)]
TQ_1:	[(1	0	0)	(0	0	0)	(0	0	0)]
TQ_2:	[(0,5	1	0)	(0	0	0)	(0	0	0)]
	"david"			"morrell"			"caine"		

and associated values of Sim_1

$Sim_1(TD_1,TQ_1) = \mathbf{0.5}$
$Sim_1(TD_2,TQ_1) = \mathbf{0.5}$
$Sim_1(TD_3,TQ_1) = \mathbf{0.5}$
$Sim_1(TD_1,TQ_2) = \mathbf{1.25}$
$Sim_1(TD_2,TQ_2) = \mathbf{0.25}$
$Sim_1(TD_3,TQ_2) = \mathbf{0.25}$

©

Since $Sim_1(TD_2, TQ_1) = Sim_1(TD_3, TQ_1) = 0.5$ we get the same result as without ***A***, but due to the paths transformation, the output also contains D_1 and $Sim_1(TD_1,TQ_1)$ is also equal to 0.5, although we work with documents of not the same structure. The highest $Sim_1(TD_1,TQ_2)$=1.25 concerns the document D_1. It is caused by the fact, that query Q_2 is a subtree of D_1. For the first query the structures

of D_2 and D_3 are similar, but their element contents match only partially ("david" vs. "david caine" and "david morrell"), i.e. with the result 0.5.

The high similarity of D_1 a Q_2 is caused also by the path transform matrix. This matrix favours documents with more detailed structure (i.e. in path of form `book⇓ author⇓ firstname` and `book⇓ author⇓ surname`). It is due to the transformation `book⇓ author⇓ firstname` 0.5 times on `book⇓ author` and `book⇓ author⇓ surname` 0.5 times on `book⇓ author`. In contrary the inverse transformation from `book⇓ author` to `book⇓ author⇓ firstname` is equal to 0 as well as the transformation `book⇓ author` to `book⇓ author⇓ surname`.

Now we describe in detail, how the transformation is processed. Let ***D*** be a matrix of dimension $m\Delta k$, where m is the number of all terms of C, k is the number of all paths in DG_C, and $w_{i,j} \subset <0,1>$. Let ***A*** be a path transform matrix of dimension k, where $a_{i,j} \subset <0,1>$ and $a_{i,i}=1$. Then a *one step transformation* is a mapping 1st_tr(***D***, ***A***) returning again a matrix of dimension $m\Delta k$, ***TD***, whose items $w_{i,j}$ are defined as

$$w_{l,i} = \max(w_{l,i}, \max_{j=1,\acute{e},k}(a_{j,i} * w_{l,j})) \qquad (2)$$

A transformation of the document matrix ***D*** by the path transform matrix ***A*** is the transitive closure of 1st_tr(***D***, ***A***).

Example 1 (continued): We will try to calculate similarities for a more complicated path transform matrix.

A_2	a	b	c	
a	1	0,2	0,2	`book⇓ author`
b	0,5	1	0	`book⇓ author⇓ firstname`
c	0,5	0	1	`book⇓ author⇓ surname`

We obtain the matrices ***TD*** and ***TQ***

	a	b	c	a	b	c	a	b	c
TD_1:	[(0,5	1	0,1)	(0,5	0,1	1)	(0	0	0)]
TD_2:	[(0,5	0,1	0,1)	(0	0	0)	(0,5	0,1	0,1)]
TD_3:	[(0,5	0,1	0,1)	(0,5	0,1	0,1)	(0	0	0)]
TQ_1:	[(1	0,2	0,2)	(0	0	0)	(0	0	0)]
TQ_2:	[(0,5	1	0,1)	(0	0	0)	(0	0	0)]
		"david"			"morrell"			"caine"	

and associated values of Sim_1.

$Sim_1(TD_1,TQ_1)$ = **0.72** $Sim_1(TD_1,TQ_2)$=**1.26**
$Sim_1(TD_2,TQ_1)$ = **0.54** $Sim_1(TD_2,TQ_2)$=**0.36**
$Sim_1(TD_3,TQ_1)$ = **0.54** $Sim_1(TD_3,TQ_2)$=**0.36**

©

Since the current form of function Sim_1 favours a little the document D_1 because of 1s in its matrix, we normalize the vectors used in definition of Sim_1. Most

approaches to IR scale usually document vectors so that they all have unit length. The length of a vector is calculated by summing the squares of all the components and taking the square root of the answer. Consequently, we obtain a score function based on the following formula:

$$Sim_2(D_i, Q) = \frac{\sum_{j=1}^{k} w_{i,l,j} * q_{l,j}}{\sum_{l=1}^{m} \sqrt{\sum_{j=1}^{k} (w_{i,l,j})^2 * \sum_{j=1}^{k} (q_{l,j})^2}}$$

Recalculating our example,

$Sim_2(TD_1, TQ_1) = \mathbf{0.62}$	$Sim_2(TD_1, TQ_2) = \mathbf{1}$
$Sim_2(TD_2, TQ_1) = \mathbf{1}$	$Sim_2(TD_2, TQ_2) = \mathbf{0.62}$
$Sim_2(TD_3, TQ_1) = \mathbf{1}$	$Sim_2(TD_3, TQ_2) = \mathbf{0.62}$

the resulted ranking of documents answered for the query Q_1 can be $\{D_2, D_3, D_1\}$. For the query Q_2 we can get $\{D_1, D_2, D_3\}$.

4. Some results and issues

Statement 1: The similarity Sim_2 is (a) irreflexive, (b) non-negative, (c) commutative, and (d) not transitive.

Proof: It is easy to verify the (c) property. The proof of (a) can be shown by example of $Sim_2(TD_3,TD_3) = 2$. In general, the result of Sim_2 can be from $<0,m>$, i.e. (b) holds. For example, it is 0 in the case, if the sets of terms with non-zero weights have empty intersection for documents D_i, D_j, $i \prod j$. Consequently, from the result of $Sim_2(D_1,D_2)$ it is not possible to decide that D_1 a D_2 are the same. The (d) property follows, e.g., from the fact that $Sim_2(D_3, D_1)+Sim_2(D_1, D_2) = 0$ { $Sim_2(D_3, TD_2) = 1$ for $D_1 = (0, 0, 1)$, $D_2 = (0, 1, 0)$, and $D_3 = (0, 1, 0)$, i.e. the triangular inequality does not hold. Thus, the similarity Sim_2 is not a metric.

Example 2: In Fig. 4 the document D_2 in collection C_2 distinguishes from D_1 and D_3 in its structure, since it uses the element `<title>` instead of `<name>`. Obviously, D_2 comes from other source. Fig. 5 shows an associated DataGuide. Since we want to work with all such documents evenly we set up the similarity between paths `book⇓ name` and `book⇓ title` as 1 in $\boldsymbol{A}$.

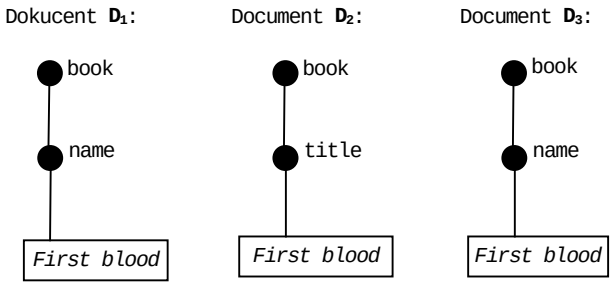
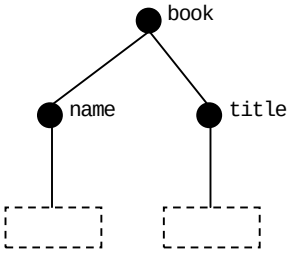


Figure 4. Collection C_2 of XML documents



paths:

```
book->title
book->name
```

Figure 5. DataGuide for C_2

The matrix ***A***

A_3	a	b
a	1	1
b	1	1

book⇓ name
book⇓ titul

and the modified document matrix based on DG_{C2}

	a	b	a	b
TD_1: [	(0,5	0,5)	(0,5	0,5)]
TD_2: [	(0,5	0,5)	(0,5	0,5)]
TD_3: [	(0,5	0,5)	(0,5	0,5)]
	"first"		"blood"	

used for the calculation of Sim_2 provide the following results:

$$Sim_2(TD_1,TD_1)=Sim_2(TD_1,TD_2)=Sim_2(TD_1,TD_3)=Sim_2(TD_2,TD_3)=2$$

Due to the transformations specified in ***A***, the similarity of a document to itself is the same for all documents. It is in accordance with our requirements. We can conclude that the impossibility to determine the sameness from the results of Sim_2 is rather contradictory.

©

Consider the example of transformations between book⇓ author and book⇓ author⇓ firstname, book⇓ author⇓ surname. We can observe that asymmetric path transform matrix favours some structures. Asymmetry of transformation in one direction and vice verse between two paths causes that the path with better transformation contributes more to the resulted similarity, i.e. documents with this structure will be favoured. For usual situations we can assume that the similarity between two paths expressed with one number is enough. Thus, the path transform matrix can be symmetric. On the other hand, a favoring some path can be beneficial as it is documented in Example 1.

We conclude with an issue how to deal with a mixed content in this approach. In the matrix model an element with a mixed content will be indexed as the union of its partial text parts. For example, the element

<x>text$_1$<z>é</z>text$_2$<y>é</y><x>

generates paths x⇓ z, x⇓ y, and x, where x refers to text$_1$ ≅ text$_2$.

5. Experiments

To use the matrix model in practice for a XML data collection *C*, it is necessary to explore

- set of terms,
- list of collection paths,

and to set up the associated path transform matrix. The first two data structures are partially generated during lexical analysis of entering the XML documents. With paths it is possible to construct a DG_C. A considerable theoretical foundation behind DataGuides can be found in [12], which proved that creating a DataGuide over a source database is equivalent to a well-studied problem concerning conversion of a nondeterministic finite automaton to a deterministic finite automaton. In our case, when the source database is a tree, this conversion takes linear time.

A computation of the transitive closure of $\boldsymbol{A}$ is relatively complex. In [13], a greedy algorithm is considered with complexity $O(k^4)$, where k is the number of paths in DG_C. In the second step we calculate the transformation of $\boldsymbol{D}$. Its complexity is $O(m*k^2)$. Observe, these calculations run only once. Assuming a naµve implementation of query evaluation, we obtain the complexity $O(m*k)$.

Our algorithm has been implemented in Java. We did a comparison between vector and matrix model. For our first experiments we used the well-known collection of Shakespeareôs plays [3] and synthetic data generated by a widely used database benchmark XBench [17]. The former dataset, C_1, contains 37 plays of the total size 7712KByte. We identified 14201 measurable terms 39 paths there. Documents of the latter collection have been sized 9722 KByte. This collection, C_2, contained 26 documents with 22 paths in DG_C and 2712 measurable terms. Since there is no difference between a document and query in our approach we used some members of collection directly as queries. As the efficiency has been not the main goal our experiments we pay a rather high price to more complex calculations in the matrix model comparing to the vector space model. Memory representations of both type of indices are not too different.

For experiments with different queries we started with "hamlet.xml" query against all plays in both models. (Fig. 6 and 7) and the following paths:

play⇓ personae⇓ pgroup⇓ persona

to

play⇓ personae⇓ persona

by constant 0.2 and

play⇓ title

to

play⇓ playsubt
by 0.5, and back by constant 0.5 too.

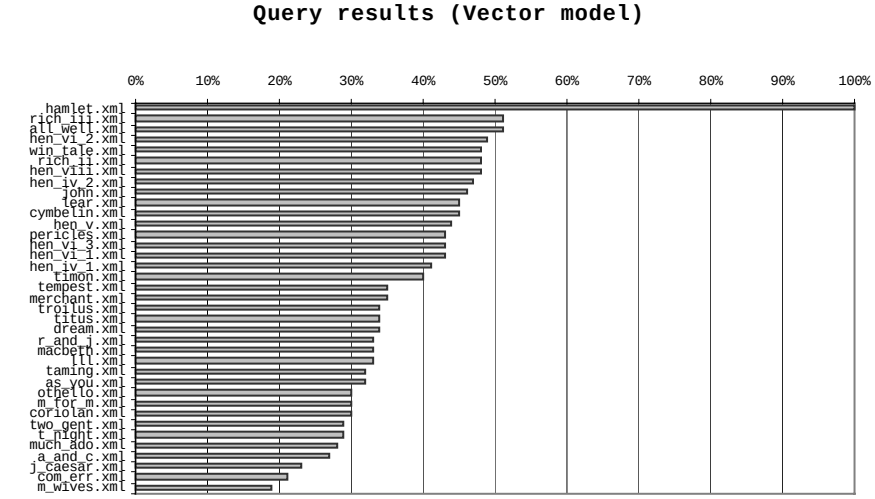


Figure 6. Result of ñhamlet.xmlò query in the vector space model

We can observe from the graphs in Fig. 6-8 that the matrix model has more balanced results, the values are less dispersed. The reason lies in the document structures that distinguish minimally for particular plays.

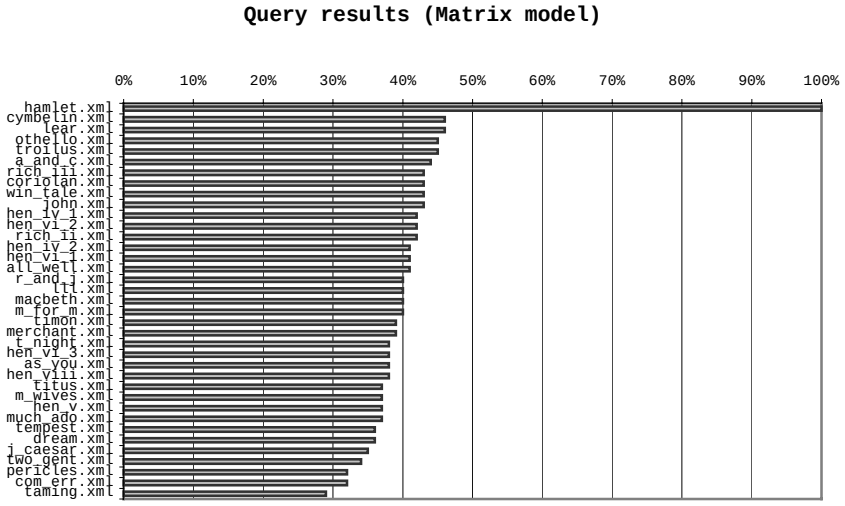


Figure 7. Result of ñhamlet.xmlò query in the matrix model

The vector space model gives some results of lower quality. For example the tragedies Othello, Mackbeth, and Romeo and Juliet are of less relevance comparing

to the comedy "A Midsummer Night's Dream". For finding out a play similar to the tragedy Hamlet it is not too good result. Othello is even under the 30%. Here the matrix model uniquely wins since all mentioned plays have the similarity higher than 40%.

Bigger differences are also observable with Julius Cesar, The Comedy of Errors and The Merry Wives of Windsor However Julius Cesar did not obtain better ranking in the matrix model too. The matrix model also gives more balanced similarities for plays about the king Henry.

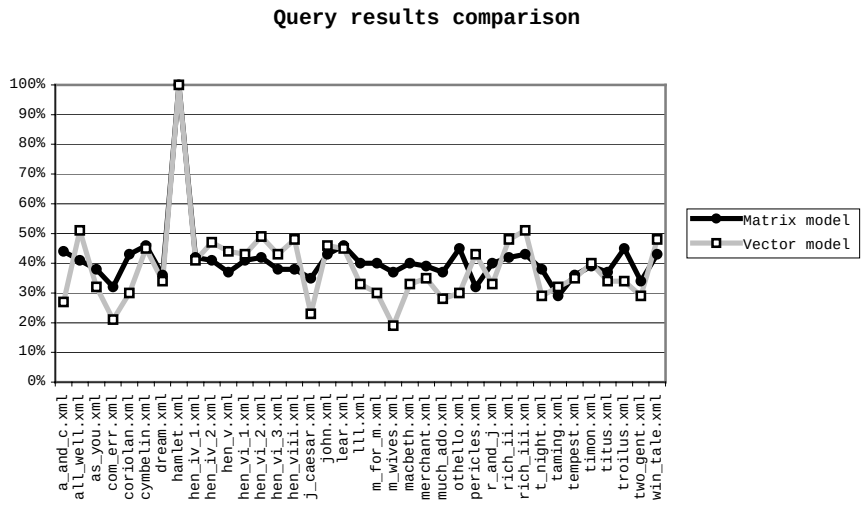


Figure 8. Comparison of the query results in the vector and matrix models

6. Conclusions

The matrix model solves problems connected with adding the structure of documents to convenient vector space model known from IR. Its goal is to enhance possibilities of integrating heterogeneous sources of XML data.

In principle, the matrix model does not distinguish between a query and a document. It is designed for finding out the similarity of two XML documents. The model can be extended for a multi-lingual information sources. Each path in the path transform matrix could be set up for a different language according to the requirements on indexing.

Similarly to other approaches based on transformations, a manual approach to tuning the *A* matrix is important. Although this could be conceived as a drawback of the method, we believe that any improving of todayôs information retrieval methods requires such approach, as only poor statistical data is not enough for achieving this goal. Obviously, the use of machine learning techniques and probabilistic estimation based on training sets of appropriate benchmarks may also be beneficial.

Since nowadays XML is increasingly used as a replacement for HTML on the Web, documents usually have XLinks or XPointers to data in other documents. Such

inter-documents links could be also added to the searching their contents. Our future work will aim to use the matrix model also in the context of such applications.

References

[1] Anh, V.N., Moffat, A.: Compression and an IR Approach to XML Retrieval. In: Proc. of the First Workshop of the Initiative for the Evaluation of XML Retrieval, Dagstuhl, Germany, December 2002.

[2] Abiteboul, S., Quass, D., McHugh, J., Widom, J., and. Wiener, J. L.: The Lorel Query Language for Semistructured Data. *International Journal on Digital Libraries*, 1(1):68ï 88, 1997.

[3] Bosak, J.: Shakespeare 2.00. *Los Altos, California*, http://www.ibiblio.org/bosak/, 1999

[4] Bourret, R.: XML and Databases. Available at http://www.rpbourret.com/xml/XMLAndDatabases.htm.

[5] Bremer, J.-M., Gertz, M.: XQuery/IR: Integrating XML Document and Data Retrieval In: Proceedings of the 5th International Workshop on the Web and Databases (WebDB), June 2002.

[6] Carmel, D., Efraty, N., Landau, G.M., Maarek, Y., Mass, Y.: An Extension of the Vector Space Model for Querying XML Documents via XML Fragments. In: Proc. of XML and Information Retrieval (Workshop) Tampere, Finland, 2002.

[7] Crouch, C.J., Apte, S., Bapat, H.: Using the Extended Vector Model for XML Retrieval. In: Proc. of INEX 2002 Workshop, 2002.

[8] Fox, E.A.: Extending the Boolean and Vector Space Models of Information Retrieval with P-Norm Queries and Multiple Concept Types. PhD thesis, Cornell University Department of Computer Science, 1983.

[9] Fuhr, N., GroGjohann, K.: XIRQL: A Query Language for Information Retrieval. In: *Proc. of ACM-SIGIR*, New Orleans, 2001, pp. 172-180.

[10] Grabs, T., Schek, H.: Generating vector spaces on-the-fly for flexible XML retrieval. In: Proc. of XML and Information Retrieval (Workshop) Tampere, Finland, 2002.

[11] Goldman, R., Widom, J.: DataGuides: Enabling Query Formulation and Optimization in Semistructured Databases. In: *Proc. of the 25th VLDB Conf.*, 1997, pp. 436-445.

[12] S. Nestorov, J. Ullman, J. Wiener, and S. Chawathe. Representative objects: Concise representations of semistructured, hierarchical data. In Proceedings of the 13th Int. Conf. on Data Engineering, Birmingham, England, April 1997.

[13] Rejlek, V.: A similarity of XML documents. Master Thesis, Dep. of Software Engineering, Charles University, 2003, (in Czech).

[14] Salton, G. (editor), *The SMART Retrieval System*, Prentice Hall, New Jersey. (1971)

[15] Extensible Markup Language (XML) 1.0., http://www.w3.org/TR/REC-xml, 1998.

[16] XML Path Language (XPath), Version 1.0, http://www.w3.org/TR/xpath

[17] Yao, B.B., ¥zsu, M.T., Keenleyside, J.: XBench - A Family of Benchmarks for XML DBMSs. *School of Computer Science University of Waterloo*, http://db.uwaterloo.ca/~ddbms/projects/xbench/index.html, 2002.

SCIENTIFIC PAPERS UNIVERSITY OF LATVIA. 2004. VOLUME 672, Datbases and Information Systems
177–191 P.

Querying Heterogeneous Distributed XML Data

Abdelsalam Almarimi

Czech Technical University, Faculty of Electrical Engineering
Karlovo nam. 13, Praha, Czech Republic
belgasem@cslab.felk.cvut.cz

Jaroslav Pokorny

Charles University, Faculty of Mathematics and Physics
Malostranske nam. 25, Praha, Czech Republic
pokorny@ksi.ms.mff.cuni.cz

Abstract. This paper describes a proposal for a system for XML data Integration and Querying via Mediation (XIQM). An XML mediation layer is introduced as a main component of XIQM. It is used as a tool for querying heterogeneous XML data sources associated with XML schemas of diverse formats. Such a tool manages two important tasks: mappings among XML schemas and XML data querying. The former is performed through a semi-automatic process that generates local and global paths. A GUI tree structure for each XML schema is constructed, which is a simple form used for assigning indices manually to match local paths to corresponding global paths. By gathering all paths with the same indices, the related local and global paths were grouped automatically, and an XML Metadata Document was constructed. An XML Query Translator for the latter task is implemented to translate a global user query into local queries by using the mappings that are defined in the XML Metadata Document.

Keywords. Distributed XML data, data integration, mediation, metadata, XML Schema

1. Introduction

Modern business often needs to combine heterogeneous data from different data sources. Tools and infrastructures for data integration are required. Recently, XML [16] has appeared as the standard for data representation and data exchange on the Web. This is mainly due to the increase in distributed heterogeneous data sources. The advantages of XML as an exchange model, such as rich expressiveness, clear notation, and extensibility, make it the best candidate for supporting the integrated data model. In this context, multiple heterogeneous data sources available on-line have increased, and have thus created a need for access to these heterogeneous data sources in a collective manner. Therefore, tools are required to mediate between XML queries and heterogeneous data sources to translate user queries into local queries.

As the importance of XML has increased, a series of standards has grown up around it, many of which were defined by the World Wide Web Consortium (W3C). For example XML Schema language [17, 18, 19] provides a notation for defining new types of XML elements and XML documents. An XML document is usually associated with a Document Type Definition (DTD) or XML schema that is used to define and constrain the syntax structure of a document. Due to the limitations of DTD, we consider XML documents and their associated XML schemas. XML with itself-describing hierarchical structure and the language XML Schema provide the flexibility and manipulative power needed to accommodate distributive and heterogeneous data. At the conceptual level, they can be visualized as hierarchical trees or graphs.

The schema integration process involves three main stages: conflicts analysis, conflicts resolution, and schemas merging. During conflicts analysis differences in the schemas are identified. In the second stage the conflicts are resolved. Finally, the schemas are merged into a single global schema using the decisions made during the previous stage. In this context, it is necessary to resolve several conflicts caused by the heterogeneity of the data sources with respect to data model, schema or schema concepts. For schema-level conflicts several classifications were proposed in the literature, e.g. [11, 15]. In contrast, integration on the instance level considers the concrete data in the sources. Here, the mapping between entities from different sources representing the same real-world objects has to be defined. The main difficulty is that the data at different sources may be represented in different formats and in incompatible ways. For example, the bibliographical databases of different publishers may use different formats of authors' or editors' names or different units of prices. Moreover, the same expression may have a different meaning, and the same meaning may be specified by different expressions.

Our system prototype called XIQM (XML data Integration and Querying via Mediation) has been built to perform the mappings among XML schemas, producing a mediation layer, which is then used to generate local queries. The mediation layer is proposed as a main component to describe the mappings between global XML schema and local heterogeneous XML schemas. It produces a uniform interface over the local XML data sources and provides the required functionality to query these sources in a uniform way. It involves two important units: the XML Metadata Document (XMD) and the Query Translator. The XMD is an XML document containing metadata, in which the mappings between global and local schemas are defined. The XML Query Translator, which is the integral part of the system, is implemented to translate a global user query into local queries by using the mappings that are defined in the XMD. Currently, we use Quilt [4] as XML query language, but we can move to XQuery language without problems

The rest of the paper is organized as follows. Section 2 introduces the related work. In section 3 we present an overview of XQIM architecture. Section 4 describes the schema integration process. XML schema parsing and constructing GUI process are introduced in Section 5. In section 6 we show the XML Metadata Document generation. Section 7 describes the query translator unit. Some query translation examples are introduced in Section 8. Finally, we summarize the paper and point out the future work.

2. Related work

In the recent years, there have been many research projects focusing on heterogeneous information integration. Most of them are based on common mediator architecture [6]. In this architecture, mediators provide a uniform user interface to views of heterogeneous data sources. They resolve the query over global concepts into subqueries over data sources. Mainly, they can be classified into structural approaches and semantic approaches.

In structural approaches, local data sources are assumed as crucial. The integration is done by providing or automatically generating a global unified schema that characterizes the underlying data sources. On the other hand, in semantic approaches, integration is obtained by sharing a common ontology among the data sources. According to the mapping direction, the approaches are classified into two categories: global-as-view and local-as-view [9]. In global-as-view approaches, each item in the global schema is defined as a view over the source schemas. In local-as-view approaches, each term in each source schema is defined as a view over the global schema. The local-as-view approach better supports a dynamic environment, where data sources can be added to the integration system without the need to restructure the global schema.

Well-known research projects and prototypes such as Garlic [8], Tsimmis [7,10], MedMaker [21], and Mix [3] are structural approaches and take a global-as-view approach. A common data model is used, e.g., OEM (Object Exchange Model) in Tsimmis and MedMaker. Mix uses XML as the data model; an XML query language XMAS was developed and used as the view definition language there. DDXMI [20] (for Distributed Database XML Metadata Interface) builds on XML Metadata Interchange. DDXMI is a master file including database information, XML path information (a path for each node starting from the root), and semantic information about XML elements and attributes. A system prototype has been built that generates a tool to do the metadata integration, producing a master DDXMI file, which is then used to generate queries to local databases from master queries. In this approach local sources were designed according to DTD definitions. Therefore, the integration process is started from the DTD parsing that is associated to each source.

Many efforts are being made to develop semantic approaches, such as RDF (Resource Description Framework) [5] and Knowledge-based Integration [2]. Several ontology languages have been developed for data and knowledge representation to assist data integration from semantic perspective such as Ontolingua [1]. F-logic [14] is employed to represent knowledge in the form of a domain map to integrate data sources at the conceptual level.

We classify our system as a structural approach and differ from the others by following the local-as-view approach. The XML Schema language is adopted instead of DTD grammar language.

3. An overview of XQIM architecture

The entire architecture of XQIM is presented in Figure 1. We assume that all database sources that are contain XML data, and each source is associated with its

XML schema definition. The main component of the system is the mediation layer, which comprises the XML Metadata Document (XMD) and the Query Translator. The XMD is an XML document containing metadata, in which the mappings between global and local schemas are defined.

The function of the query translator is rewriting a parsed global query into a subquery for each local source. The main idea is that when a *global query* over the global XML schema is posed, it is automatically translated by the Query Translator unit to subqueries, called *local queries*, which fit each local database format using the information stored in XMD.

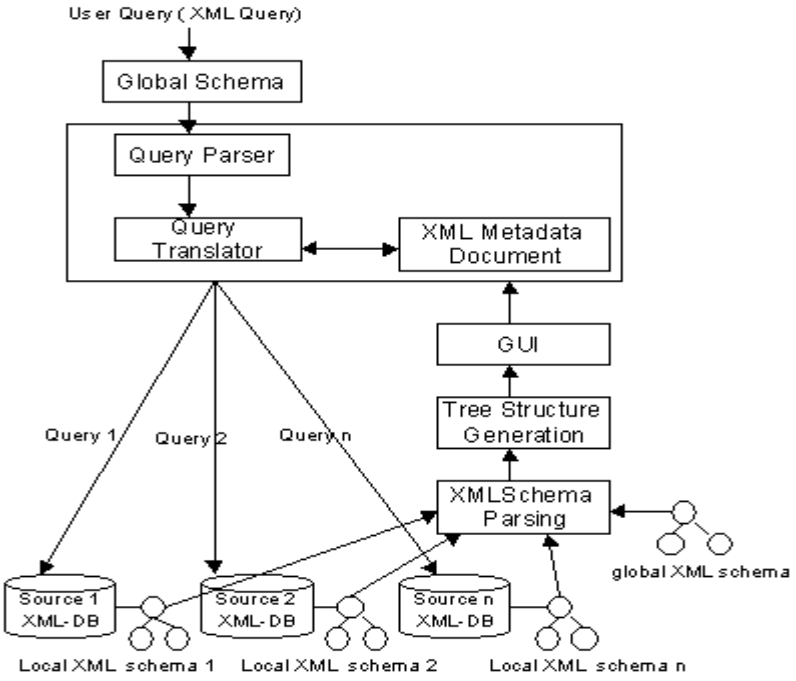


Figure 1. XQIM Architecture.

4. XML schema integration process

The integration process involves a set of data sources contain heterogeneous XML data. The integration is obtained through a virtual global XML schema that characterizes the underlying local sources. We used the strategy in which a set of local XML schemas is merged into a single global XML schema. The process starts with detecting the differences and the semantic correspondences in the schemas. Then, the conflicts are resolved and the global XML schema is defined in which all local elements are included. To clarify our approach, we introduce an example in which three publishers' database sites are used. Our objective is to create a global view over these heterogeneous sites. The publishers are Addison Wesley (AW),

Prentice Hall (PH), and Wiley. The structure of each site was studied carefully and their XML schemas were defined. For clarity we present in Figure 2 the tree structures of the global and Wiley schemas, respectively. In Figure 3 we show a part of XML schema definition for the global XML schema.

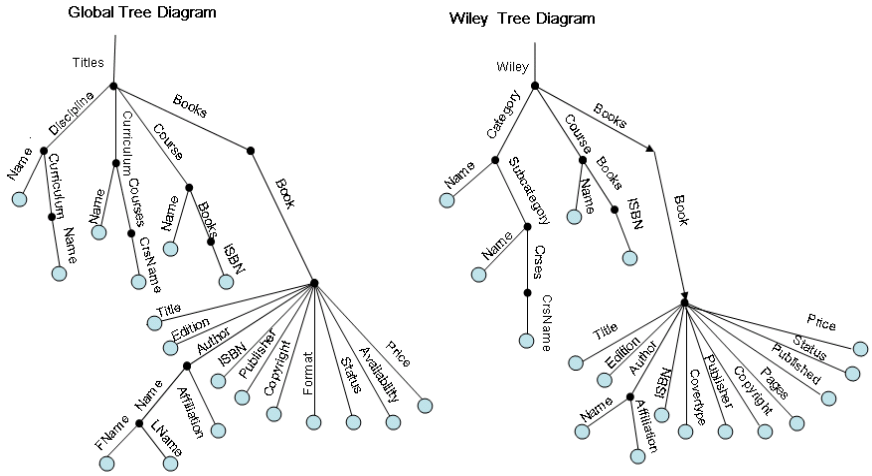


Figure 2. The global and Wiley tree structures.

```
<?xml version = "1.0" ?>
<schema xmlns = "http://www.w3.org/2001/XMLSchema"
          targetNamespace = "C:/FileLocationPath">
<element name = "Titles">
  <complexType>
    <sequence>
      <element name = "Discipline"   type = "DiscType"   maxOccurs = "unbounded"/>
      <element name = "Curriculum"  type = "CurrType"   maxOccurs = "unbounded"/>
      <element name = "Course"       type = "CourseType" maxOccurs = "unbounded"/>
      <element name = "Books"        type = "BookType"   maxOccurs = "unbounded"/>
    </sequence>
 </complexType>
</element>
<complexType name= "DiscType">
    <sequence>
      <element name =  "Name" type = "string"/>
      <element name =  "Curriculum">
        <complexType>
          <sequence>
            <element name  = "Name"  type = "string"/>
          </sequence>
        </complexType>
      </element>
    </sequence>
</complexType>
<complexType name = "CurrType">
    <sequence>
     <element name = "Name" type = "string"/>
     <element name =  "Courses">
       <complexType>
         <sequence>
           <element name  = "CrsName"  type = "string"/>
          .............
```

Figure 3. A part of the global XML schema.

5. XML schema parsing

We have used JDOM API for reading an XML schema document in memory. JDOM is a tree-based, pure Java API for parsing, creating, manipulating, and serializing XML documents. JDOM represents an XML document or XML schema document as a tree composed of elements, attributes, comments, processing instructions, text nodes, CDATA sections, and so forth. The entire tree is available at any time. JDOM itself does not include a parser. Instead it depends on a SAX [12] parser to parse documents and build JDOM models from them. Once a document has been loaded into memory, whether by creating it from scratch or by parsing it from a stream, JDOM can process the document. A JDOM tree is fully updateable. Once we have parsed a document and formed a `document` object, we need to search it to select out those parts, which we are interested in. Formally, we introduce a function

$$\text{CHILD: ELEMENTS} \Downarrow \cap(\text{ELEMENTS})$$

which assigns a multiset of child elements to each element in an XML document[1].

A GUI tool, which is a simple form, is used in XQIM to simplify the mappings between schemas. A part of a GUI is shown in Figure 6. The second column is used for assigning a unique index number for the related element paths. The third column is used to specify the functions, which needed to resolve heterogeneity conflicts.

In our example, for each source, its XML schema document is read, parsed and its root element passed with a depth of zero and a tree representation is produced as in Figure. 4. The number of spaces indicates the depth in hierarchy. This tree represents the whole XML schema document including the name and the type of each node. For example `element AW AWType` in XML Schema language is interpreted as: `element` is a defined name in the language, its value is `AW`, and its type is `AWType`, which is defined as a `complex type`.

We divided the process of XML schema parsing and GUI constructing for each source into the following steps:

1. Element names' values (exclude the name and type) are extracted and a new tree data structure *x* is constructed.
2. A unique number is given to each node of *x* to resolve naming conflicts.
3. A depth-first traversal is performed on *x*; the CHILD function is materialized. Figure 5 shows the generated function (represented by a table) for AW source. We observe that, e.g. for the node `AW 1` we obtain the associated set of its children (here represented as an array)`[Discipline 2, Curriculum 3, Course 4, Books 5]`.
4. A GUI is generated for the global schema with a sequence of indices for the global elements appeared in the second column.
5. Finally a GUI is generated for each local schema. This can be used by the designer to assign manually the indices and the required functions that is conforming the global and local element paths.

[1] In our implementation CHILD is realized by the JAVA 2 hash table feature assigning to a parent as the key its children as values.

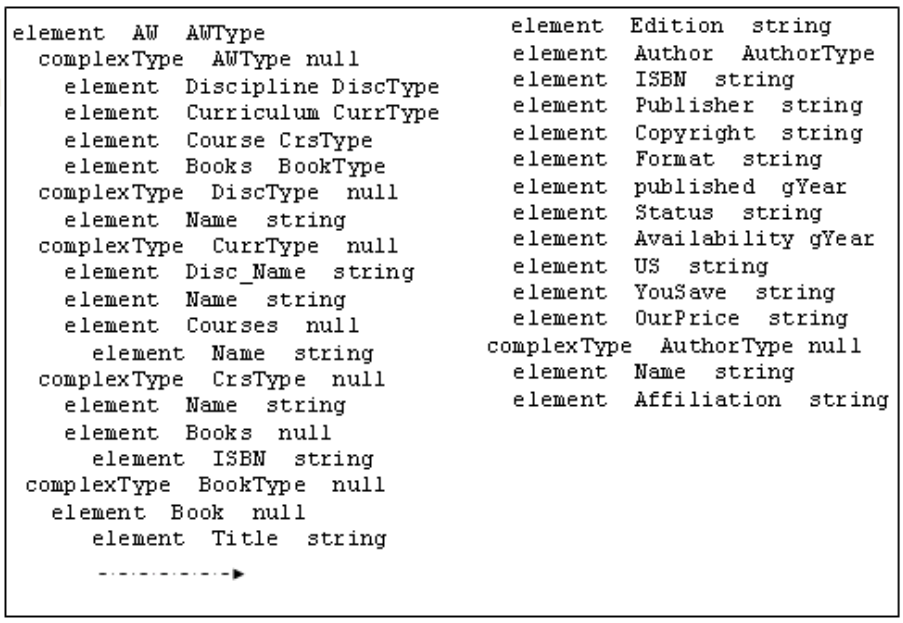


Figure 4. Parsed XML schema for the source AW.

6. The XMD generation

The XMD is an XML document containing the path information to be applied to each local source, along with identification information to be used for query translation. From Figure 8, we observe that XMD document contains the `<source>` element (for each global element path) followed by their `<destination>` elements (represent corresponding local element paths). Moreover, it contains the information about the required functions, which is represented by `<path>` element for each global element path followed by `<function>` element for each local source. The value of the `<function>` element indicates the required operation for such global element. For each source, its XML schema is parsed as explained in Section 5.

The XMD generation process comprises the following steps:

1. The generated CHILD table for each schema is traversed to generate all paths of the XML schema tree structure from the root to each element.
2. The same index number is assigned for each local element path corresponding with its global element path using the second column in the GUI that is generated in Section 5. Figure 7 shows a sample of mappings among global and local paths.
3. In the third column of the GUI either the `Null` value is specified in the case of one-to-one mapping, or the required function name is specified in the other cases.

4. By gathering the same indices of each local element with its corresponding global element, the XMD document is easily created.

```
{ Author 17     =  [ Name 28, Affiliation 29 ],
  Curriculum 3 =  [ Disc_Name 7, Name 8, Courses 9 ],
  Courses 9     =  [ Name 10 ],
  Course 4      =  [ Name 11, Books 12 ],
  AW 1          =  [ Discipline 2, Curriculum 3, Course 4, Books 5 ],
  Discipline 2  =  [ Name 6 ],
  Books 12      =  [ SBN 13 ],
  Books 5       =  [ Book 14],
  Book 14       =  [ Title 15, Edition 16, Author 17, ISBN 18,
                     Publisher 19, Copyright 20, Format 21,  published 22,
                     Status 23, Availability 24, US 25, YouSave 26, OurPrice 27 ]
}
```

Figure 5. CHILD function table for the source AW.

Books	14	Null
Book	141	Null
Title	1411	Null
Author	1412	Null
Name	14121	Null
Affiliation	14122	Null
ISBN	1413	Null
Edition	1414	Null
Publisher	1415	Null
CoverType	1416	Concat_fun
Pages	1417	Concat_fun
Copyright	1418	Null
Price	1419	Null
Submit	Reset	

Figure 6. A part of the GUI for the source Wiley.

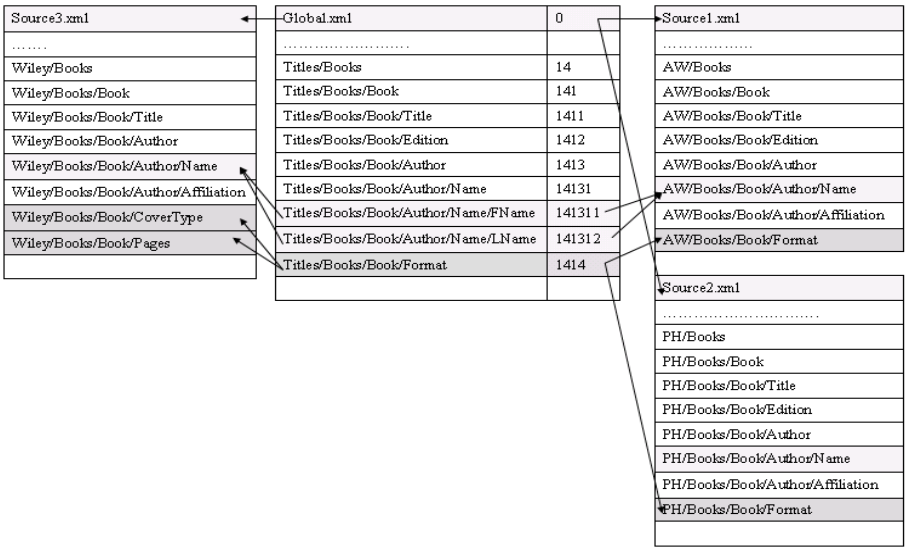


Figure 7. Sample of global and local paths mappings with index numbers.

```
<?xml version="1.0"  ?>
<Med_elem >
   <source>"global.xml"</source>
      <destination>"source1.xml"</destination>
      <destination>"source2.xml"</destination>
      <destination>"source3.xml"</destination>
   <source>Titles/Books/Book/Author/Name/FName</source>
      <destination>AW/Books/Book/Author/Name</destination>
      <destination>PH/Books/Book/Author/Name</destination>
      <destination>Wiley/Books/Book/Author/Name</destination>
   <source>Titles/Books/Book/Author/Name/LName</source>
      <destination>AW/Books/Book/Author/Name</destination>
      <destination>PH/Books/Book/Author/Name</destination>
      <destination>Wiley/Books/Book/Author/Name</destination>
   <source>Titles/Books/Book/Format</source>
      <destination>AW/Books/Book/Format</destination>
      <destination>PH/Books/Book/Format</destination>
      <destination>Wiley/Books/Book/CoverType ,
                   Wiley/Books/Book/Pages      </destination>
    -------
</Med_elem>
<Med_Func>
    --------
    <path> Titles/Books/Book/Author/Name/FName</path>
      <function>FName_fun</function>
      <function>FName_fun</function>
      <function>FName_fun</function>
    <path>Titles/Books/Book/Author/Name/LName</path>
      <function>LName_fun</function>
      <function>LName_fun</function>
      <function>LName_fun</function>
    <path>Titles/Books/Book/Forma</path>
      <function></function>
      <function></function>
      <function>Concat_fun</function>
     --------
</Med_Func>
```

Figure 8. A part of an XMD document.

7. Query translation process

We developed a method to query the distributed heterogeneous XML data sources. A query translator unit is implemented, which is an integral part of the mediation layer. Its function is the translation of the global queries, which are expressed in the language Quilt, to a set of local queries also expressed in Quilt. That is if there is a correspondence between the elements in the global and the local XML schemas. When a global user query is posed, first it is parsed, then The XMD document is read, parsed by SAX, and the number of local sources is identified. A CHILD function table *t* is constructed for the XMD, in which each `<source>` element value in XMD (global element path) is represented as a key and associated with their `<destination>` elementsô values as values (local elementsô paths). Also `<function>` elementsô values in XMD are represented in *t* as values and its corresponding `<path>` value as key.

For each element in the global query (should be a `<source>` element in XMD), if there is a non-empty value of the corresponding local elements (`<destination>` element in XMD) is non-empty, then paths in that query are replaced by paths to the `<destination>` elements to get a local query, by navigating the XMD document. Otherwise, an empty query is generated for the corresponding path in the local query that means this query cannot be applied to such local source. Each (generated) local query is sent to the corresponding local source engine, which will execute the query locally and return the result to the global query. Figure 9 shows an example of a global user query translation.

Algorithm: **Global query translation process**
Input: global XML query *q*, global XML schema, and XMD document
Output: local XML queries q_1, q_2é , q_n
Step1: **parse** *q;*
Step2: read XMD, identify the number of local sources;
Step3: **construct** CHILD function *t* for XMD; // source elements as keys and destination elements as values.
Step4**:** **for each** global element g_e in *q* **do**
materialize t;
for each source S_i in *t* having the corresponding local element l_e to g_e **do**
generate local query q_i for the first element occurrence; //only once
for each g_e whose Function element value is not `Null`
generate the required function operation
endfor
replace g_e by l_e in q_i;
endfor
endfor
Step5: **execute** the generated local queries locally.

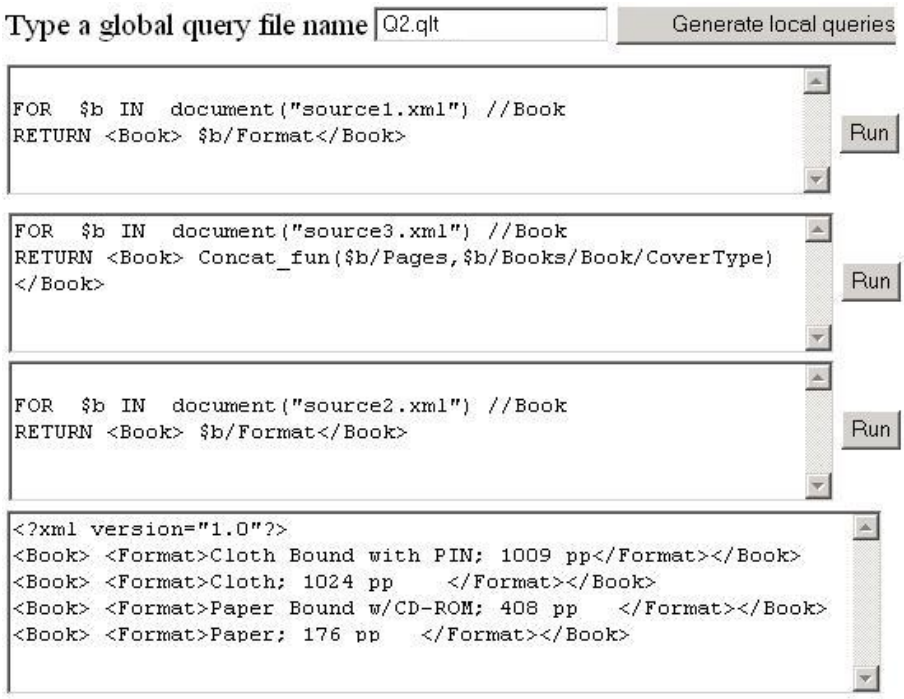


Figure 9. Example of a global query translation.

8. Examples of Query Generation

In this section we introduce examples of Quilt queries, which are provided to support our analysis. Three cardinality cases are investigated in the following subsections: one-to-one, one-to-many, and many-to-one.

8.1 One-to-One mapping example

```
Q1:  FOR  $a IN document ("global.xml") //Book
     RETURN <Book> $a/Title</Book>
```

In this case, the steps follow this order:

1. Q1 is parsed and `<Title>` element is detected and treated as a `<Source>` element in the XMD document.
2. The CHILD function is constructed for the XMD document by putting each `<source>` element path as a key and their `<destination>` element paths as values.
3. The CHILD(ó`Title`ô) function is executed to get the destination values for the `<source>` element `Title`.

4. A local query is generated for each `<destination>` element whose value is not `null`.

8.2 One-to-Many mapping query example

Q2:
```
FOR $b IN document ("global.xml")//Book
RETURN <Book>$b/Format</Book>
```

This case can happen when there is a node in the global schema mapped to many nodes in a local schema. Figure 10 below describes the mapping of this query. When running the above query, the routine work that was mentioned in the case of one-to-n is also applied. Here, we have more than one element path in local schema holds the same index number. Hence, we need to combine these paths in a one-destination element. Therefore, to answer this kind of mapping we need to provide a specific function to perform this task. For example, the global element `Format` is represented in Wiley by two different local elements: `Pages` and `CoverTypes`. The following fragment of code illustrates the desired generated local query for source Wiley (`source3.xml`).

```
FUNCTION Concat_fun($par1, $par2)
{  concat( $par1, $par2) }
FOR $b  IN  document(ñsource3.xmlò) // Book
RETURN <Book> Concat_Fun($b/Pages, $b/CoverType) </Book>
```

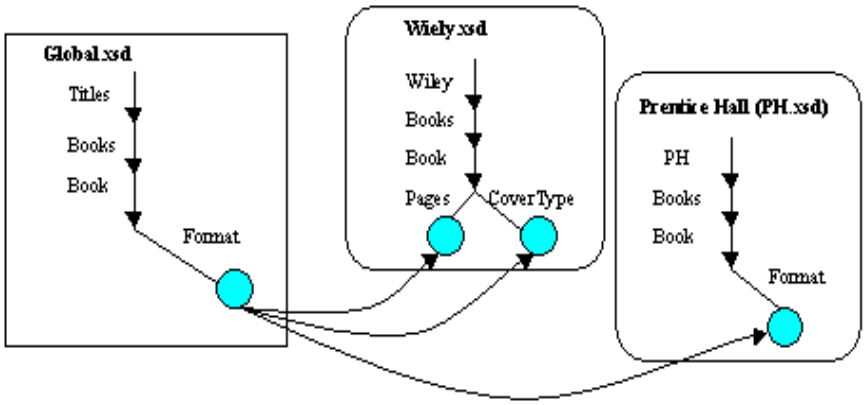


Figure 10. One-to-Many mapping example.

8.3 Many-to-One mapping query example

Q3:
```
FOR $a IN document ("global.xmlò)//Author//Name
RETURN <Author><Name> $a/LName, $a/FName </Name></Author>
```

If two or more nodes of the global schema correspond to one node in a local schema, then the node in the local schema will have more than one path. Figure 11

describes this mapping case. Here, `LName` and `FName` elements in the global schema are mapped to `Name` element in a local schema. That means we need to provide a specific function to separate the full name into a first name and last name. Also, for this kind of mapping, the same routine work that was mentioned in the subsection 7.1 is applied.

From the XMD fragment in Figure 8 we observe that two different `<source>` element values (in XMD document) have the same `<destination>` element values. In other words, we need to split the instance value of the destination element value to generate the appropriate local queries. The resulting local query for AW source as following:

```
FUNCTION  Fname_fun($par)
{  split( ñ  ñ, $par)[1] }
FUNCTION LName_fun($par)
{ document (split( ñ  ñ, $par)[2] ) }
FOR $b  IN  document(ñsource1.xmlò) // Book
RETURN <Book> FName_Fun($a/name), LName_Fun($a/name) </Book>
```

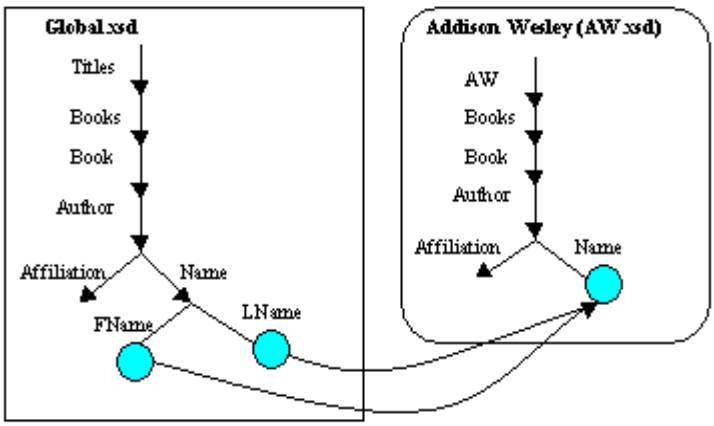


Figure 11. Many-to-One mapping example.

9. Conclusions and future work

In this paper, we have described our system for resolving structural and semantic conflicts for distributed heterogeneous XML data. We developed a mediation layer for querying heterogeneous distributed XML data sources. This layer holds two main parts: the XMD and the Query Translator. We used XML Schema language for defining the XML data sources. The mediation layer is used for describing the mappings between global and local schemas. XML schemas' trees are generated automatically, each with a GUI. Semantic discrepancies are resolved by using the GUI tool for assigning index numbers to all database elements' paths. The same

index number is assigned to nodes with the same meaning to resolve the conflicts. The same index numbers are collected to generate a global path for each element with their corresponding local element paths. Then, the XMD is generated.

Also, we have presented the second part of the mediation layer, the Query Translator. It acts to decompose global queries into a set of subqueries. A global query from an end-user is translated into local queries for XML data sources by looking up the corresponding paths in the XMD. Java 2, JDOM, JavaCC, and the Java servlet server were used as tools for the prototype implementation of this proposal.

In this work, we can only handle simple mappings among elements. In future, we aim to handle mappings that involve attributes; for example an attribute in one schema is represented as an element in another one. The integration of the local queries' results is not yet implemented. In addition, we plan to move to XQuery, instead of Quilt.

References

[1] A. Farquhar, R. Fikes and J Rice: The Ontiliqua Server: A tool for Collaborative Ontology Construction. International Journal of Human-Computer Studies, 1997, pp. 707-728.

[2] B. Ludascher, A. Gupta, and M. E. Martone: Model-based Mediation with Domain Maps. In: Proc. of ICDE, 2001, pp. 81-90.

[3] C. Baru, A. Gupta, B. Ludascher, R. Marciano, Y. Papakonstantinu, P. Velikhov, and V. Chu. XML-Based Information Mediation with MIX. In: Proceedings of the ACM SIGMOD International Conference on Management of Data, 1999, pp. 597-599.

[4] D. Chamberlin, J. Robie, and D. Florescu. Quilt: An XML Query Language for Heterogeneous Data Sources. WebDB 2000, LNCS 1987, Springer, 2001.

[5] D. Brickley, R. Guha, "Resource Description Framework (RDF) Schema Specification 1.0", W3C Candidate Recommendation, Mar 27, 2000, http://www.w3.org/TR/rdf-schema.

[6] G. Wiederhold, Mediators in the Architecture of Future Information System, IEEE Computer Magazine, Vol. 25, No. 3, March 1992, pp. 38-49.

[7] J. Ullman: Information Integration Using Logical Views. In: Proceedings of the International Conference on Database Theory, 1997, pp. 19-40.

[8] L. Haas, D. Kossman, E. Wimmers, J. Young: "Optimizing Queries across Diverse Data Sources", 23rd Int. Conf. On Very Large Databases (VLDB97), Athens, Greece, 1997, pp. 276-285.

[9] M. Lenzeini. Data Integration : A Theoretical Perspective. In Proc. Of the ACM Symposium on Principles of Database Systems (PODS), Madison, Winsconsin, USA, June 2002, pp. 233-246

[10] S. Chawathe, H. Garcia-Molina, J. Hammer, K. Ireland, Y. Papakonstantinou, J. Ullman, and J. Widom: The TSIMMIS Project: Integration of Heterogeneous Information Sources. In: Proc. of IPSJ Conference, Tokyo, Japan, October 1994, pp. 7-18.

[11] S. Spaccapietra, C. Parent, Y. Dupont: Model independent assertions for integration of heterogeneous schemas. In: VLDB Journal, Vol. 1, No. 1, pp. 81-126.

[12] SAX 1.0: The Simple API for XML. http://www.perfectxml.com/wp/3110_Chapter06/contents.htm

[13] V. K. Chaudhri, A. Farquhar, et al. OKBC: Open Knowledge Base Connectivity 2.0. Technical report KSL-98-06, Knowledge System Laboratory, Stanford, July 1997.

[14] W. May: A Rule-Based Querying and Updating Language for XML. In: Proceedings of DBPL, Springer LNCS 2397, 2001, pp. 165-181.

[15] W. Kim, J. Seo, Classifying Schematic and Data Heterogeneity in Multidatabase Systems, IEEE Computer Magazine,Vol. 24, No. 12, pp. 12-18, 1991.

[16] W3C Consortium: Extensible Markup Language (XML). http://www.w3.org/TR/2000/REC-xml

[17] W3C Consortium: XML Schema Part 0 : Primer. http://www.w3.org/TR/2001/REC-xmlschema-0-20010502/

[18] W3C Consortium: XML Schema Part 1: Structures. http://www.w3.org/TR/2001/REC-xmlschema-1-20010502/.

[19] W3C Consortium: XML Schema Part 2: Datatypes. http://www.w3.org/TR/2001/REC-xmlschema-2-20010502/.

[20] Y. Nam, J. Goguen, G. Wang, A Metadata Integration Assistant Generator for Heterogeneous Distributed Databases. In: Proc. of Confederated International Conferences DOA Irvine CA, October 2002, LNCS 2519, Springer, pp. 1332-1344.

[21] Y. Papakonstantinou, H. Garcia-Molina, J. Ullman. MedMaker: A Mediation System Based on Declarative Specifications. In: Proc. of the IEEE International Conference on Data Engineering (ICDE), New Orleans, LA, February 1996, pp. 132-141.

SCIENTIFIC PAPERS UNIVERSITY OF LATVIA. 2004. VOLUME 672, Datbases and Information Systems
192–204 P.

Evolution of Schema of XML-documents Stored in a Relational Database

Andrey Simanovsky

St Petersburg State University
Math&Mech, CS Dept.
asimanovsky@acm.org

Abstract. XML is today a standard for manipulating semistructured data. One of widely used industrial solutions, especially for systems with a fairly well defined data structure, is storing XML in a relational database, while XML queries are converted to SQL queries to the underlying relational database. A software product that produces XML "interface" to an underlying relational database commonly requires revision of XML and relational schemas with every new version of the product. Those schema and subsequent data transformations are selected and performed using *ad-hoc* algorithms. We propose a framework, namely formal evolution model, allowing semi-automatic schema transformation and data transformation for a new product version, thus discarding the necessity of *ad-hoc* algorithm design. The framework also allows a-priory estimating of query conversion performance.

Keywords. schema evolution

1 Introduction

The eXtensive Markup Language is today a de facto standard for manipulating semistructured data. This creates a set of data management challenges of storing and querying XML documents. One of widely engaged approaches is storing XML in a relational databases and providing query processor, which converts queries to XML documents to SQL queries to these underlying databases [6]. Even though industrial systems provide XML support and there are native XML databases, the relational approach dominates because it allows utilizing highly developed RDBMS technologies.

In many applications XML documents belonging to a particular domain comply with a particular schema designed for the domain. A number of algorithms of storing XML documents in relational databases, in particular those based on inlining [11], make use of the schema to provide better performance of the system.

Schemas used for particular domains tend to evolve in time, while many documents satisfying old schemas are already stored in relational database. E. g. it is a common practice for many software products to change both, XML and relational schema, with every next version of the product, whereas the transformation algorithms for old versions of the database are developed *ad-hoc*. Such changes are often driven by new functionality requirements and, thereupon, regard performance as a minor question. Our example on Figure 1 models that case.

```
DTD99:
<!ELEMENT book (booktitle, author)>
<!ELEMENT article(title, author*, contactauthor)>
<!ATTLIST contactauthor authorID IDREF IMPLIED>
<!ELEMENT monograph (title, author, editor)>
<!ELEMENT editor(monograph*)>
<!ATTLIST editor name CDATA #REQUIRED>
<!ELEMENT author (name, address)>
<!ELEMENT name(firstname?, lastname)>
DTD01:
<!ELEMENT book (booktitle, price?, author, authority*)>
<!ELEMENT authority (authname, country)>
<!ELEMENT monograph (title, author, editor)>
<!ELEMENT editor(monograph+)>
<!ATTLIST editor name CDATA #REQUIRED>
<!ELEMENT author (name, address)>
<!ELEMENT name(firstname, lastname)>
```

Figure 1. Two DTDs for books taken from [9] (DTD99) and [11] (DTD01).

We argue that schema evolution is a framework that enables partial automation of the process of schema and data transformations and allows discarding of *ad-hoc* algorithms. We present a schema evolution model for a schema of XML documents that especially takes into account relational storage and introduces guidelines for the relational schema design as well.

There are two issues that a schema evolution problem consists of: the semantics of change, i. e., the changes of database structure, and change propagation, i. e., the transformation of existing data so that it complies with the new schema. The latter question was largely investigated in recent time ([10], [13], [15]) and has no specifics for XML documents stored in a relational database as compared to a general case. In this paper we concentrate on the former issue.

The model has many in common with traditional approach to evolution of object-oriented data. However, it is less restrictive than straight-forward application of techniques used for temporal object-oriented databases (i. e. those described in [1]).

1.1 Related work

Many research projects address storing XML in relational databases and providing efficient querying to the stored data. [5] and [8] provide an overview together with approximate performance estimates of main ideas underlying the majority of techniques of storing XML documents in a relational database for both situations: when the schema of the document is known and when it is not. [6], [9], [11] propose different algorithms based on shredding (storing) an XML document into a relational database with inlining method as well as algorithms of translation of queries over XML into relational queries. [12] discusses questions of storing order information for inlining methods and other algorithms of XML document shredding. However, these works do not address the issue of schema evolution concentrating rather on the per-

formance questions. Our work utilizes the results obtained in these works considering them in the new aspect of schema evolution.

The schema evolution problem was largely investigated for temporal relational and object-oriented databases (e .g. in [1]). There were attempts to apply solutions for object-oriented databases to the evolution of schemas of XML documents. An evolution model and a classification of elementary schema transformations, with which it was suggested to describe general schema transformations, are proposed in [14]. However, presented evolution models often require multiple steps to acquire intuitively elementary schema changes. E. g. in our example swapping tags *contactauthor* and *authorid* may seem an atomic action while it would require three elementary operations to be performed on the schema if we employ general evolution model from [14].

1.2 Roadmap

In Section 2 we review the notion of a DTD-graph. In Section 3 we discuss DTD-graph factorizing. We use DTD-graph factorizing to build a lattice similar to the type lattice used when evolution of object-oriented databases is described. In Section 4 we present invariants for a factorized DTD-graph, and describe schema evolution in their terms. In Section 5 we describe elementary operations of the model and corresponding DTD transformations. In Section 6 our results are summarized and future research directions are outlined.

2 Building a DTD-graph

The process of building a DTD graph is explained in [11]. However, since its notion is slightly different in our work, and since it is essential for the understanding of the evolution model we present it in detail. A general DTD can be defined as follows:

Definition 2.1. *DTD definition*. Let C be a class of languages over an alphabet V. A DTD over V with respect to C is a mapping : $V \rightarrow C$.

Note that defining DTD as a mapping over a (necessarily) finite set allows describing DTD with a mapping table.

An example of C can be the class of regular languages, or a more sophisticated language, e. g. one used for DTDs defining XML document schemas. In the process of building a DTD-graph we consider a class of languages C_0 that can be defined with use of two constructs: a star ("$*$") and grouping ("$\ldots$"). We will call DTDs with respect to that class a simplified DTDs to distinguish them from DTDs used to describe real-life XML documents, which we will address to as regular DTDs.

The first stage of building a DTD-graph is obtaining a simplified DTD from a regular one. The process may be described as follows:

- The elements that are defined with `<!ENTITY>` tags are substituted with their definitions.
- The alphabet V is a union of tag and attribute names of the original DTD.

- Each <!ELEMENT> or <!ATTLIST> element of the DTD is considered as an entry in the mapping table that defines the simplified DTD.

- Words REQUIRED, IMPLIED etc are interpreted in terms of "$*$" and " " signs.

- Regular expression language over DTD tags used in ELEMENT tags is replaced with a language from the simplified DTDs language class using the rules presented in [11] and replacing all ' | ' with grouping.

DTD99:
<!ELEMENT book (booktitle, author)>
<!ELEMENT article(title, author*, contactauthor)>
<!ELEMENT contactauthor (authorID)>
<!ELEMENT monograph (title, author, editor)>
<!ELEMENT editor(monograph*)>
<!ELEMENT author (name, address)>
<!ELEMENT name(firstname, lastname)>
DTD01:
<!ELEMENT book (booktitle, price, author, authority*)>
<!ELEMENT authority (authname, country)>
<!ELEMENT monograph (title, author, editor)>
<!ELEMENT editor(monograph*)>
<!ELEMENT editor (name)>
<!ELEMENT author (name, address)>
<!ELEMENT name(firstname, lastname)>

Figure 2. Simplified DTDs for DTD99 and DTD01.

The simplified DTDs on Figure 2 correspond to the regular DTDs of the example described earlier.

Having a simplified DTD over an alphabet V with respect to a set of languages C_0 we define a DTD-graph as follows:

Definition 2.2. *DTD-graph*. DTD-graph is a graph (V, E) where V is the mentioned above set of attributes and $E \subseteq V \times V \times \{"*", " "\}$

Vertices of the DTD-graph denote the tags and attributes of the original DTD. Edges of the DTD-graph denote the nesting relations between tags. The "$*$"-mark appears on those edges that denote one-to-many nesting relationship. Figure 3 demonstrates a DTD-graph obtained for DTD99.

A number of algorithms (e. g., those in [12]) use similar notions of a DTD-graph extending it with additional information to provide a mapping of XML documents into relational database.

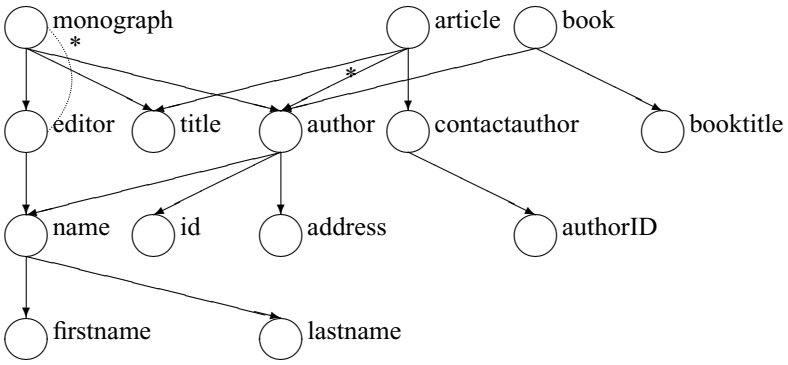


Figure 3. DTD-graph for DTD99.

3 Introducing a lattice

At first we consider a factorization (normalization) of the DTD-graph. The following notions are used.

Definition 3.1. *DTD-graph roots*. $Roots$ is an explicitly selected non-empty subset of DTD-graph ($Roots \subseteq V \ \wedge \ Roots \neq \emptyset$) vertices including all vertices without incoming edges: $\forall v \ In(v) = \emptyset \ \Rightarrow \ v \in Roots$

The roots of a DTD-graph are vertices corresponding to XML tags that can appear as outermost tags of an XML-document. We will assume that there is just one root where the existence of multiple roots is insignificant.

Definition 3.2. *Dominator*. A vertex of a DTD-graph is a dominator of a given vertex iff it appears on any path from any root to the given vertex. The domination relation is denoted as $a < b$ (denotes that a is a dominator of b).

Definition 3.3. *Immediate dominator*. Immediate dominator of the given vertex is a dominator of the given vertex, which is connected to the vertex with an edge without "$*''$-mark on it.

Note that "$*''$-mark restriction makes our notion of immediate dominator more restrictive than the generally accepted term is.

Definition 3.4. *Attribute*. Attribute is a vertex without outgoing edges.

Definition 3.5. *Factorized DTD-graph*. Factorized DTD-graph G is a 4-ple (V', E', A, α) where $V' \subseteq V^*$ and the following is true:

- factorized vertex sets are disjoint : $V_1 \neq V_2 \in V' \ \Rightarrow \ V_1 \cap V_2 = \emptyset$;
- no "$*''$-marked edges are present in a factorized vertex : $((v_1, v_2), sign) \in E \ \wedge \ \exists V_0 \mid v_1, v_2 \in V_0 \Rightarrow \ sign = \text{“ ''}$;
- sets are normalized with respect to the domination relation $<$: $v_1, v_2 \in V_0 \ \wedge \ v_1 < v_2 \ \Rightarrow \ \forall v \in In(v_1) \, v < v_2$;

- E' is a multi-set of pairs from $V' \times \{\text{“}*\text{”}, \text{“ ”}\}$;
- set of attributes A : $A = \{v \mid Out(v) = \emptyset\}$;
- vertex attributes α is a mapping : $V' \rightarrow A^*$ such that $\alpha(V_0) \subseteq V_0$.

Note that G can be created from a DTD-graph by merging vertices with their immediate dominators. G is obtained as soon as there are no more vertices to merge.

Many algorithms build relational schema on the basis of a DTD-graph. The factorization technique described largely follows the method used in [11] to construct relational schema. We do not assume that the created sets of attributes are the relations that are stored in a database, though we expect that the attributes of G correspond to attributes of the relational schema. I. e., results of the mapping α are views over the relational schema, and the stored procedures that calculate the views are stored in the database.

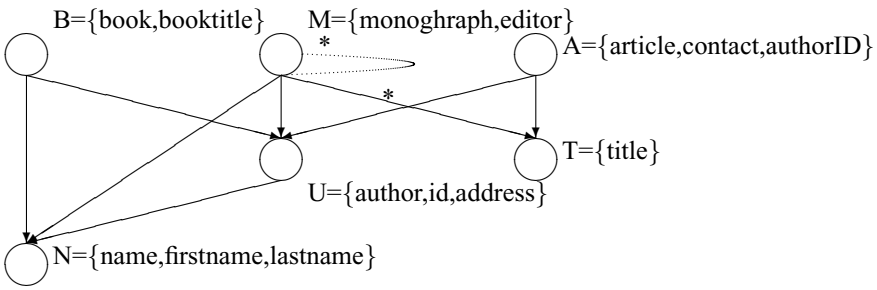


Figure 4. Factorized DTD-graph for DTD99.

A factorized DTD-graph for our working example is presented on Figure 4.

4 Factorized DTD-graph invariants

We suggest describing schema evolution model through a set of invariants applied to G. A natural way to formalize restrictions on the graph would be requiring that selected sets of vertices or attributes should (not) appear on each (or any, especially if an original DTD contained ‘—’ rather than grouping) (acyclic) path from any root to the given vertex. These restrictions are invariants (axioms) that any accepted schema should comply with.

In the case of acyclic DTD-graph a possible set of invariants was presented for object-oriented databases in [1]. This set of axioms is presented on Figure 5. It can be applied with slight terminology changes. The following notions is used.

Definition 4.1. *Immediate predecessors*. The hierarchy on V' is defined by the sets of immediate predecessors $P(t)$ for each vertex t. Immediate predecessors are vertices that explicitly include vertex t and do not include t indirectly (through a transitive inclusion).

1. Closure: $\forall t \in V'\ P_e(t) \subseteq V'$
2. Acyclicity: $\forall t \in V'\ t \notin \cup\alpha_x(PL(x), P(t))$
3. Rootedness: $\exists T \in V'\ \forall t \in V'\ T \in PL(t) \wedge P_e(T) = \emptyset$
4. Pointedness: $\exists \perp V'\ \forall t \in V'\ t \in PL(\perp)$
5. Immediate predecessors:
   $\forall t \in V'\ P(t) = P_e(t) - \cup\alpha_x(PL(x) \cap P_e(t) - \{x\}, P_e(t))$
6. Predecessors graph: $\forall t \in V'\ PL(t) = \cup\alpha_x(PL(x), P(t)) \cup \{t\}$
7. Interface: $\forall t \in V'\ I(t) = N(t) \cup H(t)$
8. Nativeness: $\forall t \in V'\ N(t) = N_e(t) - H(t)$
9. Inheritance: $\forall t \in V'\ H(t) = \cup\alpha_x(I(x), P(t))$

Figure 5. Axiom set for acyclic case.

Definition 4.2. *Essential predecessors*. Essential predecessors $P_e(t)$ are explicitly specified sets of vertices. It is required that $P(t) \subset P_e(t)$.

Definition 4.3. *Vertex subhierarchy*. Vertex subhierarchy $PL(t)$ is a sub-graph of G, which vertex set consists of vertices, from which t is reachable.

Definition 4.4. *Native attributes*. Native attributes $N(t)$ are a set of attributes defined in vertex t.

Definition 4.5. *Inherited attributes*. Inherited attributes $H(t)$ are the union of attributes of all its predecessors.

Definition 4.6. *Essential attributes*. Essential attributes $N_e(t)$ are explicitly specified set of attributes. It is required that $N(t) \subset N_e(t)$.

Definition 4.7. *Interface*. Interface $I(t)$ is the union of its inherited and native attributes.

$N_e(t)$ denotes attributes (i. e. leaf tags or attributes of original DTD), which values are essential for working with a vertex of G, P_e denotes essential vertices (i. e. multiple occurences of a tag in original DTD).

Existence of root T and terminating vertex $\perp$ can be replaced with existence of final sets of vertices with the same properties. That would not significantly alter any results, so we retain the form of the corresponding axioms as they were stated in [1].

Vertices with single incoming edge are allowed in the evolution model. In that case we assume that G has a forward-edge ending in a vertex with a single incoming edge or the edge is marked with "$*$". If several possibilities for a given vertex exist we allow any — evolution model does not specify, which one to select.

To handle cyclic factorized DTD-graphs we introduce several additional notions.

Definition 4.8. *Immediate descendants*. The reverse hierarchy on V' is defined by the sets of immediate descendants $D(t)$ for each vertex t. Immediate descendants are vertices that explicitly are included into vertex t and are not included into t indirectly (through a transitive inclusion).

Definition 4.9. *Vertex reverse subhierarchy*. Vertex reverse subhierarchy $DL(t)$ is a sub-graph of G, which vertex set consists of vertices reachable from t.

Definition 4.10. *Essential descendants*. Essential descendants $D_e(t)$ are explicitly specified sets of vertices. It is required that $D(t) \subset D_e(t)$.

1. Closure: $\forall t \in V'\ P_e(t), D_e(t) \subseteq V'$
2. Rootedness: $\exists T \in V'\ \forall t \in V'\ T \in PL(t) \wedge P_e(T) = \emptyset$
3. Pointedness: $\exists \perp V'\ \forall t \in V'\ t \in PL(\perp)$
4. Immediate predecessors:
   $\forall t \in V'\ P(t) = P_e(t) - \cup\alpha_x(PL(x) \cap P_e(t) \cap (PL(t) - DL(t)) - \{x\}, P_e(t)) -$
   $\cup\alpha_x(PL(x) \cap P_e(t) \cap (PL(t) \cap DL(t)) - \{x\}, P_e(t)) - \cup\alpha_x((PL(x) - P(x)) \cap$
   $(PL(t) - DL(t)) \cap P_e(t), PL(x) \cap DL(x) - \{x\})$
5. Predecessors graph: $\forall t \in V'\ PL(t) = \cup\alpha_x(PL(x), P(t)) \cup \{t\}$
6. Immediate descendants:
   $\forall t \in V'\ D(t) = D_e(t) - \cup\alpha_x(DL(x) \cap D_e(t) \cap (DL(t) - PL(t)) -$
   $\{x\}, D_e(t)) - \cup\alpha_x(DL(x) \cap D_e(t) \cap (DL(t) \cap PL(t)) - \{x\}, D_e(t)) -$
   $\cup\alpha_x((DL(x) - D(x)) \cap (DL(t) - PL(t)) \cap D_e(t), DL(x) \cap PL(x) - \{x\})$
7. Descendants graph: $\forall t \in V'\ DL(t) = \cup\alpha_x(DL(x), D(t)) \cup \{t\}$
8. Interface: $\forall t \in V'\ I(t) = N(t) \cup H(t)$
9. Nativeness: $\forall t \in V'\ N(t) = N_e(t) - H(t)$
10. Inheritance: $\forall t \in V'\ H(t) = \cup\alpha_x(I(x), P(t))$

Figure 6. Axiom set for general case.

The axiom set for a general case is shown on Figure 6.

We introduce new axioms for controlling $D(t)$ and $DL(t)$ sets and modify axiom 4. The idea of modification is to avoid transitive inclusion to be passed through recursive vertices inclusion. The new axiom 4 states that $P(t)$ consists of those vertices from $P_e(t)$ that:

- do not lie in the area marked 2 (vertices of $P_e(t)$ that are both predecessors and descendants of t) on Figure 7 and transitively include t through a vertex from area 2,

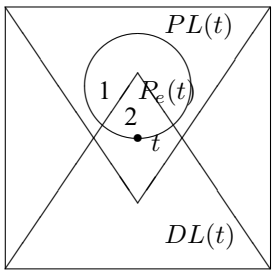


Figure 7. Understanding general case immediate predecessors axiom.

- do not lie in the area marked 1 (vertices of $P_e(t)$ that are not descendants of t) on Figure 7 and transitively include t through a vertex from area 1,
- do not lie in area 1 and transitively include t through a vertex from area 2.

The calculation of $N(t)$, $I(t)$ and $H(t)$ sets requires calculation of $PL(t)$ set. Provided that initially all sets except essential ones are empty $PL(t)$ can be calculated by iterative application of the axioms. The following claim states that the set will have the same contents regardless of axiom application order.

Claim 4.1. *$PL(t)$ set value does not depend on calculation order and its calculation requires at most $Card(V'^*)$ applications of the predecessors graph axiom.*
Proof. Due to lack of space only a sketch of proof is given. $PL(t)$ calculation through application of axioms can be regarded as an iterative algorithm (see [3]) on G with a distributive function PL defined over a semilattice V'^* with operation of strict set inclusion. Since V'^* is a finite set the semilattice complies the finite chains condition (each chain contains at most $Card(V'^*)$ elements). Thus, we can apply theorem from [3], which states that $PL(t)$ calculation process is finite and does not depend on calculation order.
The PL set can be calculated by applying the following calculation algorithm:

$\forall t P(t) := P_e(t)$;
$\forall t D(t) := D_e(t)$;
$\forall t PL(t) := DL(t) := \{t\}$;
`while` ($\exists t$ | one of the axioms does not hold for t)
Recalculate $PL(t)$ applying axioms 4 – 7 to t.

The `while` loop runs at most number of attributes by number of vertices times. The worst case graph for the algorithm is a bidirectional list with attributes in each vertex. ■

5 Elementary operations

Now when we have a set of invariants that establish requirements for G we consider a set of elementary operations of the evolution model. Each elementary operation should retain invariants defined in Section 4 if they were present in an initial schema.

We introduce eight elementary operations: adding/ removing an edge, adding/ removing an attribute, adding/ removing a vertex and splitting/ merging a vertex. Each of these operations has an equivalent operation on initial DTD. We describe under what conditions the evolution model accepts an operation for a given schema, and what changes occur in the model when the operation is performed.

- `AddAttribute`. Attribute a (the one being added) of vertex t is included into $N_e(t)$. The sets N_e, N, H are recalculated for the vertex t and vertices reachable from t. Schema designer (or an external algorithm) may include this attribute into N_e sets of some successors of vertex t. The equivalent operation on simplified DTD is adding `<!ELEMENT A>` and including it in an element T: `<!ELEMENT T(A,..)>`.

- `RemoveAttribute`. Attribute a (the one being deleted) of vertex t is excluded from $N_e(t)$, the sets N_e, N, H are recalculated for the vertex t and vertices reachable from t. Note, if $t \in P(s) \wedge a \in Ne(s)$, then according to axioms 8 and 9 attribute a is included into $N(s)$. The equivalent operation on simplified DTD is removing element `<!ELEMENT A>` and excluding it from an element T: `<!ELEMENT T(A,..)>`.

- `AddEdge`. Let edge (t, s) is added. s is added into $P_e(t)$ and t is added into $D_e(s)$. Expressions dependent on $P_e(t)$ and $D_e(s)$ are recalculated. Note, that s is added into $P(t)$ if there is no other path from s to t (same for $D(s)$). The equivalent operation on simplified DTD is including element `<!ELEMENT S>` into element T: `<!ELEMENT T(S,..)>`.

- `RemoveEdge`. This operation is complex, it may cause generation of new edges in the schema according to axioms 4 and 6. New edges will start in some predecessors of the end vertex of a deleted edge and end in its successors. Provided we remove edge (t, s), s is deleted from $P_e(t)$. All expressions dependent on $P_e(t)$ are recalculated. If axiom 2 is violated then the operation is rejected by the system. (Alternatively, the vertex t may be included into root vertex T, the change can be achieved in two stages: adding T into $P_e(t)$, and RemoveEdge for the edge (s, t). Similarly, axiom 3 for vertex s may be violated. However, if $s \in P_e(t)$, then s is added into the graph). Analogous operations are performed with $D_e(s)$. In simplest case this operation equivalent is excluding element S from element T: `<!ELEMENT T(S,..)>` is replaced with `<!ELEMENT T(...)>`.

- `AddVertex`. Provided vertex t is added. t is included into V'. The sets $P_e(t)$ and $D_e(s)$ are to be defined by schema designer (or an external algorithm), to generate incoming edges of t. t is added into $P_e(s)$ to satisfy axiom 4. Basically, $P_e(t) = T$, and changes are made using modification AddAttribute. In case a new root is added the operation is equivalent to introducing new `<!ELEMENT T(...)>`, where ... contains an old root.

- `RemoveVertex`. It is a complex modification. First, the vertex t is to be removed with the edges starting and ending in it. Second, a number of edges

from its predecessors to its successors may be added. Third, attributes of vertex t, that are essential for its successors should migrate properly. The operation is implemented in three stages: applying RemoveAttribute to all attributes of t, applying RemoveEdge to all outcoming edges, and applying RemoveEdge for all incoming edges. In the case of root removal the operation is equivalent to deleting a root element `<!ELEMENT T(...)>`, where ... contains a new root(s).

- `MergeVertex`. Vertices being merged must be connected by an edge. Merged vertex P_e, D_e and N_e sets are unions of P_e, D_e and N_e sets of vertices being merged with exclusion of themselves. In P_e sets of reachable amd D_e set of reaching vertices occurrences of merged vertices are replaced with occurrence of merged one. A merge equivalent is either replacing "$*$"-inclusion with ordinary one: `<!ELEMENT T(S*)>` is replaced with `<!ELEMENT T(S)>`, or removing inclusion of S into a third element V: `<!ELEMENT V(..,S)>` $\rightarrow$ `<!ELEMENT V(...)>`.

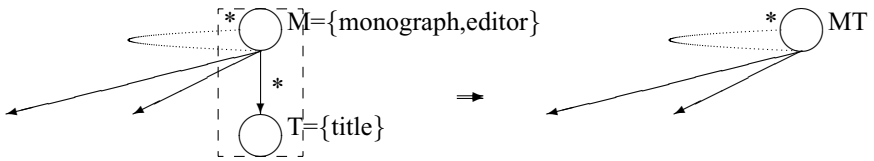


Figure 8. Merge vertex sample.

- `SplitVertex`. P_e, D_e and N_e sets of split vertex are separated into two disjoint sets each. In P_e and N_e sets of reachable vertices occurrences of split vertex are replaced with one or both of the created without violating $P(t) \subset P_e(t)$ and $D(t) \subset D_e(t)$ properties. A split equivalent is either replacing ordinary inclusion with a "$*$"-inclusion, or introducing inclusion of S into a third element V: `<!ELEMENT V(...)>` $\rightarrow$ `<!ELEMENT V(..,S)>`.

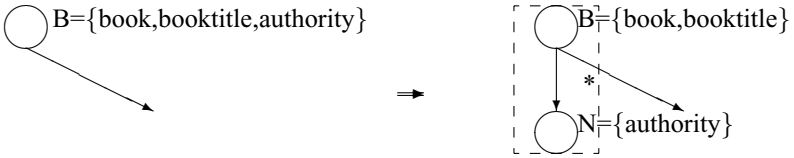


Figure 9. Split vertex sample.

The evolution from DTD99 to DTD01 for the working example may have the following sequence:

- MergeVertex(monograph,title)
- RemoveVertex(article)
  - RemoveAttribute(article,authorID)

  - RemoveAttribute(article)
  - RemoveEdge(article,author)
  - RemoveEdge(article,author)

- SplitVertex(book,authority,{authority},book)
- AddAttribute(authority,country)
- RemoveAttributeauthority
- AddAttribute(authority,authname)
- AddAAttribute(book,bestseller)
- AddAttribute(book,price)

The ability to apply one or the other elementary operation depends on the restrictions implied by the model axioms. In the evolution process the P_e, D_e and N_e (i. e. essential) sets of the model may need to be changed. The cost of an elementary operation is defined by the number of essential sets affected by it. The cost of application of a sequence of elementary operations is the sum of their costs. The preferred schema for a new version is one that requires less costly operation sequence to obtain it.

6 Conclusions

In this paper we presented a framework for schema evolution of relational database-based systems with XML interface. Through definition of P_e, D_e and N_e sets of our model it becomes possible to ensure that evolved schemas designed to answer new functional requests while still conform to both, domain-based and performance-oriented restrictions.

Our future plans include designing a more flexible way to meet performance-oriented requirements than employing inline shredding technique. We also explore ways to extend the evolution model to control more XML DTD and XML-Schema features.

References

[1] R. J. Peters, M. T. Ozsu. An Axiomatic Model of Dynamic Schema Evolution in Object-base Systems. ACM Transactions on Database Systems, 22(1), pp. 75–114, 1997.

[2] I. A. Goralwalla, D. Szafron, M. T. Szu. Managing Schema Evolution Using a Temporal Object Model. In proceedings of the 16 International Conference on Conceptual Modeling (ER'97), 1997.

[3] M. Yamamoto, K. Takahashi, M. Hagiya, S. Nishizaki, T. Tamai. Formalization of graph search algorithms and its applications. In proceedings of Theorem Proving in Higher Order Logics (TPHOLs'98), LNCS, vol. 1479, pp. 479–496. Springer-Verlag, 1998.

[4] S. Y. Lee, M.-L. Lee, T. W. Ling, L. A. Kalinichenko. Designing Good Semi-Structured Databases and Conceptual Modeling. In proceedings of International Conference on Conceptual Modeling / the Entity Relationship Approach, pp. 131–145, 1999.

[5] D. Florescu, D. Kossman. A performance evaluation of alternative mapping schemes for storing XML data in a relational database. Technical Report 3684, INRIA, 1999.

[6] J. Shanmugasundaram, H. Gang, K. Tufte, C. Zhang, D. J. DeWitt, J. F. Naughton. Relational databases for querying XML documents: Limitations and opportunities. In proceedings of 25th International Conference on Very Large Data Bases (VLDB'99), Edinburgh, Scotland, pp. 302–304, 1999.

[7] D. Florescu, D. Kossmann. Storing and Querying XML Data Using an RDBMS. IEEE Data Engineering Bulletin, 22(3), pp. 27–34, 1999.

[8] A. Deutsch, M. Fernandez, D. Suciu. Storing semistructured data with STORED. Proceedings of the 1999 ACM SIGMOD International conference on mamagement of data. pp. 431–442, 1999.

[9] Y. Men-hin, A. W.-C. Fu. From XML to Relational Database. In proceedings of 8th International Workshop on Knowledge Representation meets Databases, KRDB'2001, Rome, 2001.

[10] I. Tatarinov, Z. Ives, A. Y. Halevy, D. S. Weld. Updating XML. In SIGMOD, pp. 413–424, 2001.

[11] J. Shanmugasundaram, E. J. Shekita, J. Kiernan, R. Krishnamurthy, S. Vigilas, J. F. Naughton, I. Tatarinov. A General Techniques for Querying XML Documents using a Relational Database System. SIGMOD Record, vol. 30(3), pp. 20–26, 2001.

[12] I. Tatarinov, S. D. Viglas, K. Beyer, J. Shanmugasundaram, E. Shekita, C. Zhang. Storing and Querying Ordered XML Using a Relational Database System. International. In proceedings of the 2002 ACM SIGMOD international conference on Management of data table of contents, Madison, Wisconsin, pp. 204–215, 2002.

[13] B. Kane, H. Su, E. A. Rundensteiner. Consistently updating XML documents using incremental constraint query checks. In Web Information and Data Management (WIDM'02), pp. 1–8, 2002.

[14] S. Coox. Axiomatization of Schema Evolution in XML Databases. In Programming (3), pp. 1-9, 2003.

[15] Y. Papakomstantinou, V. Vianu. Incremental validation of XML documents. In ICDT, 2003.

DATA MINING AND DATA WAREHOUSING

SCIENTIFIC PAPERS UNIVERSITY OF LATVIA. 2004. VOLUME 672, Datbases and Information Systems
207–217 P.

Application of Data Mining Methods for Bad Debt Recovery in the Healthcare Industry

Jozef Zurada[1], Subhash Lonial[2]

Department of Computer Information Systems[1]
Department of Marketing[2]
College of Business and Public Administration
University of Louisville
Louisville, KY 40292, USA
{jmzura01,lonial}@louisville.edu

Abstract. The healthcare industry, specifically hospitals and clinical organizations, are often plagued by unpaid bills and collection agency fees. Therefore, in recent years the industry has started to apply data mining tools to reduce bad-debt balance. This paper compares the effectiveness of five such tools - neural networks, decision trees, logistic regression, memory-based reasoning, and the ensemble model in recovering bad debts. The data analysis and evaluation of the performance of the models are based on a fairly large unbalanced data sample provided by a healthcare company, in which cases with recovered bad debts are underrepresented. Computer simulation shows that the neural network, logistic regression, and the combined model produced the best classification accuracy. More thorough interpretation of the results is obtained by analyzing the response and receiver operating characteristic charts. We used the models to score all "unknown" cases, which were not pursued by a company. The best model classified about 34.8% of these cases into "good" cases. (This may potentially bring a company the total of about $423,000 - 58% in the additional recovered income.) To collect bad debts more effectively, we recommend that a company first deploy and use the models, before it refers unrecovered cases to a collection agency.

Keywords. Healthcare industry, Bad-debt recovery, Data mining methods

1. Introduction

The healthcare industry, specifically hospitals and clinical organizations, are often plagued by unpaid bills, collection agency fees and outstanding medical testing costs. As a matter of fact, hospitals typically end up paying 30% to 50% of recovered bad-debt revenue to outside collection agencies [14]. Galloro [4] reported that Nashville-based HCA's provision for bad debt rose to 10.3% of its revenue for the third quarter, compared to 8.3% of revenue in the same quarter the previous year. Pesce [9] argues that

hospitals should secure the funding for and invest in modern information technology to reduce bad-debts, which along with other factors such as billing errors, insurance underpayments and inability to collect accurate patient and payer information throughout delivery of care, account for 13% of a hospitals' lost revenue each year. The matter of recovering bad debts has become serious and has even involved hospitals suing patients. A review of court records and interviews with hospital trade groups, collection attorneys and consumer advocates shows that hospitals in several states of the U.S. are beginning to use harsh measures to collect debts and secure the arrest and even imprisonment of patients who miss court hearings related to their healthcare debts [8].

Predicting whether a particular customer is likely to repay a healthcare debt is an inherently complex and unstructured process. What makes this process especially difficult in the healthcare context is the hospital's inability to obtain detailed financial information concerning the patients. Due to moral and practical constraints, the hospital has to perform the necessary medical services on credit. Given all the difficulties of bad-debt recovery, the healthcare institution that provided data for this paper successfully recovered bad debts from only about seven percent (7.3%) of its total customers, even though the healthcare institution must have used some initial scoring model to target "good" customers who were likely to repay bad debts. Some of debt defaults may be attributed to unforeseen events (i.e. divorce, death, loss of employment) or be governed by factors that may be difficult or impossible to see in the attributes of the consumer (i.e. stability of marriage, general health, job stability). Any improvement in making a reliable distinction between those who are likely to repay the debt and those who are not would allow the healthcare institution write off the debts which are unlikely to be paid off, and hence save collection expenses.

The volume and complexity of raw data inherent in bad debt recovery can be handled by several knowledge discovery and data mining tools. Knowledge discovery is defined as the process of identifying valid, novel, and potentially useful patterns, rules, relationships, rare events, correlations, and deviations in data [6]. This process relies on well-established technologies, such as machine learning, pattern recognition, statistics, neural networks, fuzzy logic, evolutionary computing, database theory, artificial intelligence, and high performance computing to find relevant knowledge in very large databases. The knowledge discovery process is typically composed of the following phases: understanding the overall problem domain; obtaining a data set; cleaning, preprocessing, transforming, and reducing data; applying data mining tools; interpreting mined patterns; and consolidating and implementing discovered knowledge.

Data mining, which is an important phase in the knowledge discovery process, uses a number of analytical tools: discriminant analysis, neural networks, decision trees, fuzzy logic and sets, rough sets, genetic algorithms, association rules, and *k*-nearest neighbor (or memory-based reasoning) which are suitable for the tasks of classification, prediction, clustering, summarization, aggregation, and optimization. Classification and prediction are the two tasks that we deal with in this paper. These are the most common and perhaps the most straightforward data mining tasks. Classification consists of examining the features of a newly presented object and assigning it to one of a

predefined set of classes or outcomes ("debt recovered" or "debt unrecovered"). A data mining model employing one of the data mining tools must be trained using pre-classified examples. The goal is to build a model that will be able to accurately classify new data based on the outcomes and the interrelation of many discrete variables (debt amount, injury code, patient age etc.) contained in the training set.

This paper examines and compares the effectiveness of four data mining techniques (neural networks, decision trees, logistic regression, memory-based reasoning) and the ensemble model in recovering bad debts. The target/dependent variable in a fairly large data set (comprised of the training, validation, and test subsets) provided by a healthcare company represents the following three classes: 1 - "good" customers (those who repaid the debt), 2 - "bad" customers (those who defaulted), and 3 - "unknown" customers (those who were not pursued). The number of "good" customers is vastly underrepresented in the data set. To build the models, we only used cases representing "good" and "bad" customers, rejecting all "unknown" cases. Computer simulation shows that on the test set, the logistic regression model, neural network model, and the ensemble model (combines logistic regression, neural network, and decision tree) produces the best overall classification accuracy rates, and the decision tree is the best in predicting "good" customers. More subtle and meaningful interpretation of the results are obtained by analyzing the response and receiver operating characteristic charts. After building and testing the models, we used them to score 2,896 "unknown" cases, which had not been pursued by a company. The neural network model classified 1008 (34.8%) of these cases into "good" cases. This may potentially bring a company about an additional $423,000 in recovered income. In addition, our models provide the patient financial service department a list of "unknown" patients sorted from the most likely to pay the bill to the least likely to pay the bill.

The paper is organized as follows. Section 2 reviews the recent literature first in more recent business data mining applications and then in bad debt recovery applications. Section 3 describes the data sample, whereas section 4 presents some experiments and simulation results. Finally, section 5 concludes the paper and makes recommendations for future work.

2. Literature review

There is ample evidence in the literature that in cases where nonlinearities and complexities among variables are present, data mining tools such as neural networks, genetic algorithms, decision trees, fuzzy logic, rough sets, and memory-based reasoning often provide better classification accuracy rates than common statistical techniques such as regression analysis and discriminant analysis.

In more recent papers, Jagielska *et al*. [7] used the credit approval data set to investigate the performance of neural networks, fuzzy logic, genetic algorithms, rule induction software, and rough sets when applied to automated knowledge acquisition for classification problems. They concluded that the genetic/fuzzy approach compared more

favorably with the neuro/fuzzy and rough set approaches. Piramuthu [10] analyzed the beneficial aspects of using both neural networks and nuerofuzzy systems for credit-risk evaluation decisions. Neural networks performed significantly better than neurofuzzy systems in terms of classification accuracy, on both training as well as testing data. West [15] investigated the credit scoring accuracy of five neural network architectures and compared them to traditional statistical methods. Using two real world data sets and testing the models using 10-fold cross-validation, the author found that among neural architectures the mixture-of-experts and radial basis function did best, whereas among the traditional methods regression analysis was the most accurate. Thomas [13] surveyed the techniques for forecasting financial risk of lending to consumers. Glorfeld and Hardgrave [5] presented a comprehensive and systematic approach to developing an optimal architecture of a neural network model for evaluating the creditworthiness of commercial loan applications. The neural network developed using their architecture was capable of correctly classifying 75% of loan applicants and was superior to neural networks developed using simple heuristics. Yang *et al*. [16] examined the application of neural networks to an early warning system for loan risk assessment. Finally, Zurada [18] investigated data mining techniques for loan-granting decisions and predicted default rates on consumer loans.

In this paper we apply data mining tools to data-driven marketing applications. In a typical application, solicitations are mailed with the aim of achieving a certain result. Some examples include (1) credit card applications, (2) insurance policies, (3) service contracts, and (4) bad-debt recovery. Regardless of the specific application maximizing the response rate is usually the desired objective.

For example, VISA International's use of neural networks to detect fraudulent credit card transactions provided savings estimated to be $40 million over a six month period [1]. An English company has applied neural networks in direct marketing to identify the characteristics of people most likely to respond to a direct mailing campaign. The effort was worth 40,000 new customers, equivalent to $500,000 savings in mailing costs. American Express Co. has deployed neural networks in three projects. One involves a character recognition system, another is for direct mail prospects, and the third is for portfolio management trading support system [2]. In a pilot study to detect fraud conducted at American Express, neural networks provided an improvement of 3% over the previously used logistic regression models [11].

The healthcare industry has been focused on ways to reduce bad-debt balance for the last several years. In one of the earlier studies, Zollinger *et al*. [17] identified a sample of 985 patients classified as bad debt and charity cases from 28 Indiana hospitals. They built a multiple regression model and found that several institutional variables such as total hospital charge and the total hospital revenue and patient variables such as marital status, gender, diagnoses, insurance status, employment status, and discharge status were significant factors in recovering unpaid hospital bills. A similar study was performed by Buczko [3] who analyzed data on charges assigned to charity care and bad debt for 82 short-stay hospitals in Washington. The study confirmed that uncompensated

care has become a major issue in hospital finance as the number of uninsured persons has increased and hospital revenues have declined.

In one of the most recent articles, Veletsos [14] described the predictive modeling software (IBM Intelligent Miner and DB2) used for bad-debt recovery implemented by the IBM Company for the Florida Hospital at Orlando. The final study was completed in 2003 and included approximately 2,400 patients. The model is based on a variety of data variables, including credit factors, demographic information and previous organizational payment patterns. The model provides the patient financial service department a list of patients sorted from the most likely to pay the bill to the least likely to pay the bill. The model yielded approximately $200,000 in savings. In our study, which is an extension of the approach discussed by [14], we use a larger and unbalanced sample of patients and fewer variables to test the effectiveness of the five data mining models for recovering bad-debts.

3. The data sample

The healthcare company, which is the subject of this study, relied on only four factors to determine whether the bad debt was recoverable: (1) Patient Age (PA), (2) Patient Gender (PG), (3) Injury Diagnosis Code (IDC), and (4) Dollar Amount of the Claim (DAC). Over one quarter the company identified 6,319 new cases with an outstanding balance. The dependent variable Status represented groups 1, 2, and 3 containing "good", "bad", and "unknown" cases, respectively. After eliminating cases which had at least one missing value, we obtained a data set containing 6180 observations unequally divided into 449 "good" cases (group 1); 2,835 "bad" cases (group 2); and 2896 "unknown" cases (group 3).

To learn more about the distribution of the variables within the data set and to find out whether any transformation of the variables is needed, we performed a simple bivariate exploratory data analysis. For the DAC variable, the average dollar amounts of the recovered cases (group 1); not recovered (group 2); and not pursued claims (group 3) are $1,052; $417; and $254; respectively. Furthermore, the total amounts for the DAC variable for each of the 3 groups are $472,461; $1,182,350; and $734,188; respectively. Thus it appears that a company used common sense and some procedure, which allowed it to target the patients with larger debts and ignore those with smaller debts. We also performed additional transformation for the three variables. To improve the distribution of the DAC variable and obtain better prediction results, we computed and used log(DAC) instead of DAC. Furthermore, to decrease the dimensionality of the data set (the number of distinct values in the data set), we grouped the PA variable into several bins. Finally, we recorded a nominal independent variable IDC into 24 categories, which were then converted to 23 dummy variables.

4. The experiments and simulation results

We employed SAS Enterprise Miner (EM) (www.sas.com) [12] to build an initial model and a final model. The initial model was trained, validated, and tested on an unbalanced sample which contained 3,284 cases. The test data set was used to check the performance of the initial model. The initial model's overall classification accuracy was excellent and the classification rate of "bad" cases was almost perfect (100%). However, the classification rate of "good" cases was poor because "good" cases were vastly underrepresented in the training data sample, and the model classified the majority of "good" cases as "bad" cases. As a result, the initial model was found unacceptable.

In the final model, we performed stratified sampling to balance the data sample, by randomly selecting 449 "bad" cases from 2835 "bad" cases and matching them with all 449 "good" cases. After sampling, the data sample contained 898 cases equally divided between 449 "good" cases and 449 "bad" cases. This data set was divided into training, validation and test sets, each containing 450 (50%), 224 (25%), and 224 cases (25%), respectively, evenly representing the 2 groups. The first two subsets were used to build the model, whereas the test subset was used to check the performance of the model. It was assumed that detecting "good" cases was the target event.

Table 1. The test set correct classification accuracy rates for the 5 methods used. (It shows the percentage and number of test cases classified correctly.)

	Decision Tree	Neural Network	Logistic Regression	Memory-based Reasoning	Ensemble Model
"Overall"	67.9% (152/224)	72.3% (162/224)	75.0% (168/224)	61.2% (137/224)	73.7% (165/224)
"Good"	75.0% (84/112)	67.9% (76/112)	71.4% (80/112)	72.3% (81/112)	71.4% (80/112)
"Bad"	60.7% (68/112)	76.8% (86/112)	78.6% (88/112)	50.0% (56/112)	75.9% (85/112)

Table 1 compares the test set classification accuracy of decision trees, neural networks, logistic regression, memory-based reasoning, and the ensemble model for one of several computer simulations. (We ran computer simulation for a random selection of several different sets of 449 "bad" cases. Each of them was matched with the same 449 "good" cases. We also ran the experiments for several different training, validation, and test subsets. All the above scenarios yielded similar classification accuracy rates across the five models.) The ensemble model combines the best three models (neural network, logistic regression, and decision tree) by averaging the posterior probabilities of the response variable Status. It is clear that in the overall classification accuracy, logistic regression, ensemble model, and neural network outperform the other models. The decision tree, however, does the best job in classifying "good" cases. The classification accuracy rates obtained seem to be excellent if one considers the fact that the healthcare

institution that provided data for this paper successfully recovered bad debts from only about 449 of its 6180 total patients (7.3%).

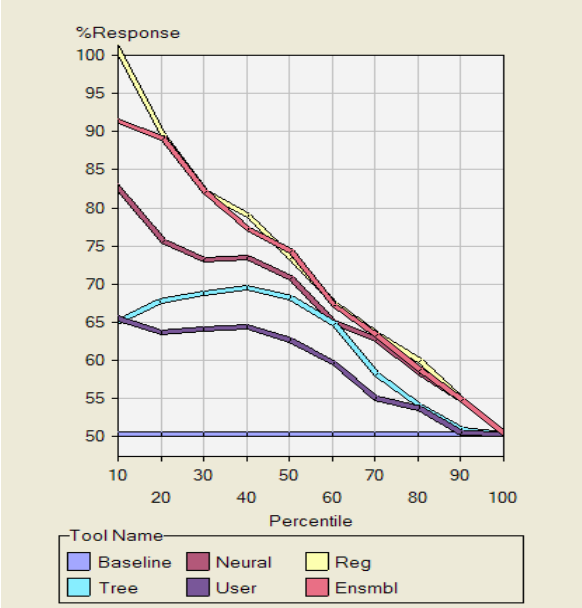


Figure 1. The test set cumulative percent response chart for the five methods. Target event: 1 ("good" cases).

The performance of the five models, however, can be best and more thoroughly evaluated by using a combination of cumulative and non-cumulative response and the receiver operating characteristic (ROC) charts. A cumulative response chart shows the overall strength of the models. To properly interpret a cumulative percent response chart, one needs to understand how the chart is constructed. In the chart (Fig. 1), a respondent is defined as an individual from whom payment is recovered ("good", Status=1). For each individual, the fitted models predict the probability that the individual will repay the debt. The observations are sorted by the predicted probability of response from the highest probability of response to the lowest probability of response, and then grouped into ordered bins, each containing approximately 10% of the data. Using the target variable Status, one can count the percentage of actual responders in each bin. If the model is effective, the proportion of individuals with the event level being modeled (in this example, those who will repay dues) will be relatively high in bins in which the predicted probability of response is high. The chart shows the percentage of respondents in the top 10%, top 20%, and so on. In a chart, the response rate for each decile of the score includes all of the responses for the deciles above it.

One sees (Fig. 1) that the 5 models predict that between 65% and 100% of the respondents in the first 10% decile will repay dues. The logistic regression, ensemble, and neural network models outperform the other 2 models with the 100%, 92%, and

83% repay rate, respectively. The regression model needs to be chosen if a company intends to target only 10% of the patients. The ensemble model, however, predicts that 75% of the respondents in the 5 first deciles (50% percentile) will repay dues, and it is slightly better than the regression model (74%) and neural network model (71%). The decision tree and memory-based reasoning (denoted as User) models undoubtedly performs worst in all deciles. The horizontal response line represents the baseline rate (approximately 50%) for comparison purposes, which is an estimate of the percentage of payers that one would expect if one were to take a random sample.

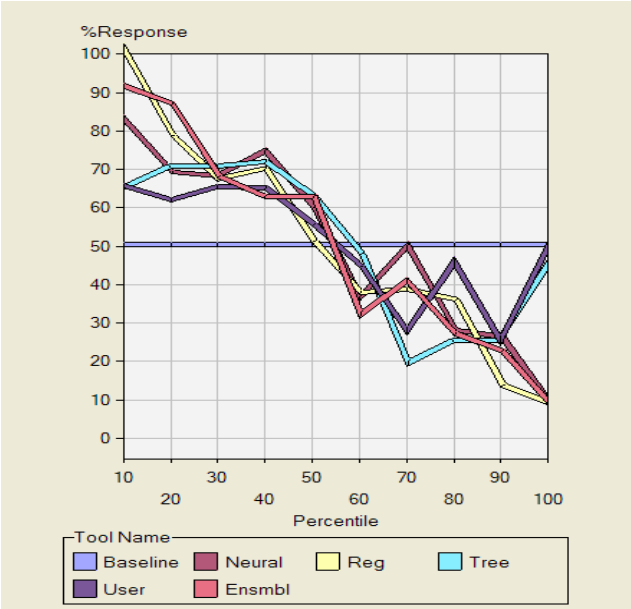


Figure 2. The test set non-cumulative % response chart for the five methods. Target event: 1 ("good" cases).

To analyze the performance of the models at each decile (strata) of the score, it is necessary to examine a non-cumulative percent response chart, which shows the percentage of "good" respondents in each decile. Fig. 2 shows that the percentage of "good" respondents for the neural network model and the ensemble model is the highest (76%) and the lowest (62%), respectively, in the 4th decile. The curves for all 5 models decline significantly between the 5th decile and the 6th decile and drop below the baseline for the 6th through the 10th decile. Below the baseline, the models become counterproductive and actually represent "bad" respondents.

The ROC charts display the global measure of the predictive accuracy of the models. They display the sensitivity on the vertical axis against 1-specificity on the horizontal axis of a classifier for a range of cutoffs. Sensitivity is a measure of accuracy for predicting events that is equal to the true positive divided by total actual positive. 1-specificity is a measure of accuracy for predicting nonevents that is equal to the true

negative divided by total actual negative. Each point on the curves represents a cutoff probability. Points closer to the upper-right corner correspond to low cutoff probabilities. Points in the lower left correspond to higher cutoff probabilities. The extreme points (1,1) and (0,0) represent no-data rules where all cases are classified into class 1 or class 0, respectively. The performance quality of the models is demonstrated by the degree to which the ROC curves push upward and to the left. The curves will always lie above the 45∀line. The area between the curves and the line provides a quantitative performance measure called the Gini index. This area will range from 50, for a worthless model, to 100, for a perfect classifier.

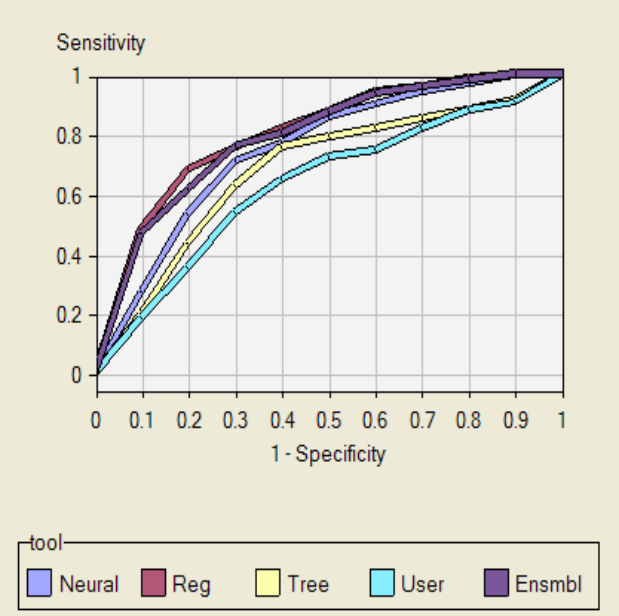


Figure 3. The test set receiving operating characteristics (ROC) chart for the five methods. Target event: 1 "good" cases.

The ROC chart (Fig. 3) indicates that the predictive power all 3 models (neural network, regression analysis, and combined model) stand out and appear to be better at predicting "good" respondents than the decision tree and memory-base reasoning models.

Finally, the best three models (logistic regression, neural network, and ensemble model) were used to score how many out of the 2896 "unknown" cases would be classified as "good" cases, bringing a company additional revenue. The results summarized in Table 2 are striking. For example, the neural network model classifies 1008 of all "unknown" cases as "good" cases (34.8%), potentially yielding the amount of $422,861 in additional revenue. This amount constitutes about (58%) of the total

amount of $734,188 for not pursued cases. The results can be explained by the fact that the models learned how to target patients with larger debts.

Table 2. The results from the Score node.

Tool	Logistic Regression	Ensemble Model	Neural Network
Number and percentage of cases classified as 1 (”good”) - cutoff probability ∅0.5	886/2896 (30.6%)	910/2896 (31.4%)	1008/2896 (34.8%)
The additional amount potentially retrieved [in US $]	$374,185	$382,148	$422,861

5. Conclusion

The paper compares the effectiveness of neural networks, decision trees, logistic regression, memory-based reasoning, and the ensemble model in recovering bad debts. The data analysis and evaluation of the performance of the various models is based on a fairly large unbalanced data sample provided by a healthcare company, in which cases with recovered bad debts are underrepresented. Computer simulation shows that the logistic regression model, neural network model, and the combined model produced the best overall classification accuracy, and the decision tree was the best in classifying “good” cases. More subtle and meaningful interpretation of the results is obtained by analyzing the response and ROC charts. We used the models to score the “unknown” cases, which were not pursued by a company. The neural network model classified more “unknown” cases into “good” cases than any other remaining models. This may potentially bring a company the additional recovered income. In addition, our models can provide the patient financial service department a list of patients sorted from the most likely to pay the bill to the least likely to pay the bill. To collect bad debts more effectively, we recommend that a company first deploy and use the models, before it turns over unrecovered cases to a collection agency.

Although the results obtained from this study could be generalized, one of its limitations may be a small number of independent variables used for prediction. In future work, we plan to (1) include for analysis more input variables including insurance employment and discharge status, if they become available to the company; (2) consider the cost profit matrix; and (3) adjust the results by prior probabilities/conditions to avoid a potential bias in the results.

References

[1] Anonymous. Visa Stamps on Fraud. *International Journal of Retail and Distribution Management*. Vol. 23, Iss. 11, pg. 1, 1995.

[2] Berry, J., A Potent New Tool for Selling: Database Marketing. *Business Week*. Iss. 3388, pg. 56, 1995.

[3] Buczko, W. Factors Affecting Charity Care and Bad Debt Charges in Washington Hospitals. *Hospital and Health Services Administration*. Vol. 39, Iss. 2, 179-191, 1994.

[4] Galloro, V. Bad News on Bad Debt. *Modern Healthcare*. Vol. 33, Iss. 43, pg. 8, Oct 2003.

[5] Glorfeld, L.W., Hardgrave, B.C. An Improved Method for Developing Neural Networks: The Case of Evaluating Commercial Loan Credit Worthiness. *Computer & Operations Research*. Vol. 23, No. 10, 933-944, 2000.

[6] Fayyad, U.S., Piatetsky-Shapiro, G., Smyth, P., Uthurusamy, R. (Eds.). *Advances in Knowledge Discovery and Data Mining*. AAAI Press / The MIT Press, Menlo Park, California, USA, 1996.

[7] Jagielska, I., Matthews, C., Whitfort, T. An Investigation into the Application of Neural Networks, Fuzzy Logic, Genetic Algorithms, and Rough Sets to Automated Knowledge Acquisition for Classification Problems. *Neurocomputing*. Vol. 24, 37-54, 1999.

[8] Lagnado, L. Hospitals Try Extreme Measures to Collect their Overdue Debts. *The Wall Street Journal*. Oct 30, 2003.

[9] Pesce, J. Satnching Hospitals' Financial Hemorrhage with Information Technology. *Health Management Technology*. Vol. 24, Iss. 8, pg.12, Aug 2003.

[10] Piramuthu, S. Financial Credit-Risk Evaluation with Neural and Neurofuzzy Systems. *European Journal of Operational Research*. Vol. 112, 310-321, 1999.

[11] Punch, L., 1994, "When Big Brother Goes to Far", *Credit Card Management*. Vol. 7, Iss. 7, pg. 22.

[12] SAS Institute. Data Mining Using Enterprise Miner Software: A Case Study Approach, 1st edition, http://www.sas.com, 1999.

[13] Thomas, L.C. A Survey of Credit and Behavioral Scoring: Forecasting Financial Risk of Lending to Consumers. *International Journal of Forecasting*. Vol. 16, 149-172, 2000.

[14] Veletsos, A. Getting to the Bottom of Hospital Finance. *Health Management Technology*. Vol. 24, Iss. 8; pg. 30, 2003.

[15] West, D. Neural Network Credit Scoring Models. *Computers & Operations Research*. Vol. 27, 1131-1152, 2000.

[16] Yang, B., Li, L.X., Ji, H., Xu, J. An Early Warning System for Loan Risk Assessment Using Artificial Neural Networks. *Knowledge-Based Systems*. Vol. 14, 303-306, 2001.

[17] Zollinger, Terrell W., Sawyell, Robert M. Jr., Chu, David K. W., Ziegert, Andrea, Woods, John R., LaBov, Dean. A Determination of Institutional and Patient Factors Affecting Uncompensated Hospital Care. *Hospital & Health Services Administration*. Chicago, 36(2), 243-256, 1991.

[18] Zurada, J., Zurada, M. Data Mining Techniques in Predicting Default Rates on Customer Loans. *Review of Business Information Systems*. Vol. 6, No. 3, pp. 65-83, 2002.

SCIENTIFIC PAPERS UNIVERSITY OF LATVIA. 2004. VOLUME 672, Datbases and Information Systems
218–231 P.

Storage Structures for Sharing Data in Multiversion Data Warehouse [1]

Bartosz Bębel[*], Robert Wrembel[*], Bogdan Czejdo[**]

[*]Institute of Computing Science, Poznań University of Technology
ul. Piotrowo 3A, Poznań, Poland
{Bartosz.Bebel, Robert.Wrembel}@cs.put.poznan.pl
[**]Loyola University, New Orleans, USA

Abstract. A data warehouse (DW) is used for the integration of data coming from external heterogeneous data sources and making them available for analytical processing, decision making, as well as data mining. The structure and content of a DW not only reflects a real world, i.e. data stored in a DW come from real production systems, but also a DW and its tools may be used for predicting trends and simulating alternative business scenarios (the what-if analysis). External data sources usually evolve in time that requires evolution of existing data warehouses. Traditional DW systems have a limitation that they are not capable of supporting any dynamics in their structure and content. For such applications as well as for the what-if analysis, a multiversion DW seem to be more appropriate. In such a DW, each DW version describes a schema and data at certain period of time or in a given business scenario. In many cases (e.g., the what-if analysis) two consecutive DW versions may contain identical sets of data. In such a case one set of data is shared by these DW versions, in order to reduce storage overhead. In this paper we propose data version storage structure, that allow sharing data between several DW versions.

Keywords. Data warehouse, schema change operations, multiversion data warehouse, data sharing.

1. Introduction

A data warehouse (DW) integrates autonomous and heterogeneous external data sources (EDSs) in order to provide an information for analytical processing, decision making, and data mining tools. Operational data, produced by OLTP (On-Line Transaction Processing) applications are periodically loaded into a DW, previously being cleaned, integrated, and often summarized. Then the data are processed by OLAP (On-Line Analytical Processing) applications in order to discover trends, anomalies, patterns of behavior, in order to predict future business trends, and to support pertinent business decisions. The subjects of analysis are called *facts* and they are described by *dimensions* that set up a context for facts.

1 This work is partially supported by the grant no. 4 T11C 019 23 from the Polish State Committee for Scientific Research (KBN), Poland

External data sources are autonomous, i.e., they may evolve in time independently of each other and independently of a DW that integrates them [22]. Changes in EDSs can be categorized as: (1) **content changes**, i.e. insert/update/delete data, and (2) changes to the structure of data, further called **schema changes**, i.e. add/modify/drop an attribute or a table. The system has to ensure the correct propagation of these changes to a DW, i.e. the structure and content of a DW must be correctly adjusted. The DW schema adjustments can be done in two different ways, namely schema evolution [5] and schema versioning [21, 19].

The first approach consists in updating the schema and transferring the data from an old schema into a new schema. Only the current version of the schema is present. In contrast, the second approach keeps track of the history of all versions of a schema. Versioning can be done implicitly by temporal extension or explicitly by physically storing different versions of a schema.

The process of good decision making often requires forecasting future business behavior, based on present and historical data as well as on assumptions made by decision makers. This kind of data processing is called the *what-if analysis*. In this analysis, a decision maker simulates in a DW changes in a real world, creates a virtual possible scenarios, and explores them with OLAP queries. To this end, a DW must provide means of creating various DW alternatives, represented by different DW versions. A DW capable of managing its various versions will be further called a multiversion data warehouse (MVDW). In order to analyze data stored in a MVDW, new analytical tools and extended query language is required.

Commercial DW systems and OLAP tools existing on the market (e.g. Oracle9i, Oracle Express Server, IBM DB2, Sybase Adaptative Server Enterprise, Ingres DecisionBase OLAP Server, NCR Teradata Hyperion Essbase OLAP Server) support neither managing changes of a DW structure, nor the what-if analysis functionality, nor querying multiversion data. The exception is SAP Business Warehouse, that is capable of handling only simple changes in dimension data. Oracle's what-if analysis allows to create only the simplest hypothetical rankings of records.

Our approach and contribution. In our approach, changes in EDSs are handled in versions of a data warehouse. A DW version is composed of a schema version and a data version. A DW administrator can group several schema changes and then apply them to a new DW version. Managing versions of a DW allows us to: (1) clearly separate various structures and contents of a DW corresponding to various periods of time; (2) create and manage various alternative virtual business scenarios required for the what-if analysis; (3) run queries addressing a particular DW version or addressing several versions and compare various factors computed in those versions often without the need of excessive transformations of data, unlike in the case of temporal approaches (e.g., [8, 9]).

In cases where two consecutive DW versions differ only by the structure of dimensions, data stored in fact tables are often the same, i.e. are not directly affected by dimension structural changes. In such cases, an identical set of fact data is shared by these DW versions, in order to reduce storage overhead. In this paper we contribute by presenting a data version storage structure, called bitmap storage, that allows sharing data between several DW versions. In our prototype system data sharing technique consists in physically storing in a given DW version only those

data that were changed in a given version or were newly inserted to this version. Other data, common to a parent and its child versions are stored only in the parent version and are shared by its child versions. For the data sharing purpose, every record, in a fact or in a level table, has attached the information about all DW versions this record belongs to. At the implementation level, the information about all versions a given record belongs to is represented in the set of bitmaps, where one bitmap represents one DW version.

Paper organization. The rest of this paper is organized as follows. Section 2 presents basic definitions in the field of DW technology. Section 3 discusses existing approaches to handling changes in the structure of a DW. Section 4 overviews our concept of a multiversion DW and discusses possible changes in a DW schema and dimension instance structure. Section 5 discusses our approach to sharing data and preliminary experimental results on query processing efficiency. Finally, Section 6 summarizes and concludes the paper.

2. Basic Definitions

A DW takes advantage of a multidimensional data model [10, 12, 13, 18, 23] with **facts** representing elementary information units in a multidimensional cube. A fact contains quantifying values, called **measures**, which are the subjects of analysis. Examples of measures include: number of items sold, income, turnover, etc. Measures depend on a context set up by **dimensions**.

Based on the definitions given in [23], a **dimension schema** is defined as a tuple (Dname, Lev, Attr, Ω, LA), where Dname is the name of a dimension, Lev represents a finite set of levels containing a special level l_{All}. Attr respresents a finite set of attributes and LA is a function that for a given attribute a_i returns a level l_i the attribute belongs to. Every dimension level l_i has associated a domain, denoted as $dom(l_i)$; the values in this domain are called **level instances**. For level l_{All} the following is true: $dom(l_{All})=\{all\}$. Ω is a partial order on levels with a unique bottom level l_{Bott} and a unique top level l_{All}, such that for a given level l_i: $l_{Bott}\ \Omega\ l_i\ \Omega\ l_{All}$. Moreover, for given two levels l_o and l_p where $l_o\ \Omega l_p$ there is no level l_j such that $l_o\ \Omega l_j\ \Omega l_p$.

An example of a hierarchical dimension is *Geography* (cf. Figure 1a), with *Countries* at the top, that are composed of *Regions*, that in turn are composed of *Cities*. *Countries*, *Regions*, and *Cities* constitute levels in this dimension.

A **dimension instance** of dimension D_i is composed of hierarchically assigned instances of levels in D_i, where the hierarchy of level instances is set up by the hierarchy of levels. Formally, a dimension instance of D_i is defined as a tuple (Val_i, F_i). Val_i consists of pairwise disjoint sets $V_{l1} \cong V_{li} \cong \ldots \cong V_{lm}$: $\&_{k\ in\ (1,\ \ldots,\ m)}\ V_{lk} \geq dom(l_k)$, where l_k is a level in D_i. F_i is the set of functions that for two given levels l_o and l_p in D_i: $l_o\ \Omega l_p$ connect instances of l_o to instances of l_p such that an instance of l_o is connected to exactly one instance of l_p.

Examples of the instances of level *Countries* may include: *Great Britain* and *France*. Examples of level *Regions* may include: *Scotland*, *Wales*, and *Bretagne*. Examples of *Cities* level may include: *Edinbourgh*, *Glasgow*, *Cardiff*, *Swansea*, *Brest*, and *Rennes*. An example of the *Geography* dimension instance is shown in

Figure 1b. Arrows between level instances represent functions in F_i. The values of dimension instances and hierarchies they form constitute a **dimension instance structure**.

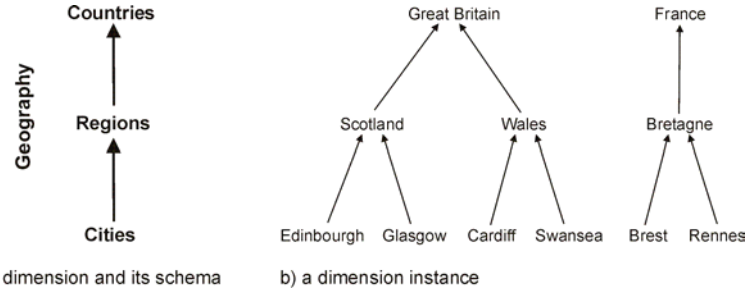


Figure 1. A dimension schema and its instance

Multidimensional data model can be implemented either in *MOLAP* (multidimensional OLAP) servers or in *ROLAP* (relational OLAP) servers. In the former case, data are stored in n-dimensional arrays. In the latter case, n-dimensional data are stored in the set of relational tables. Some of the tables represent dimensions, and are called **dimension level tables**, while others store values of measures, and are called **fact tables**. In a sequel, we will focus our discussion on ROLAP implementation of a DW.

Example 1. As a leading example let us consider a DW schema presented in Figure 2. It is used for analyzing sale of cosmetics by shops in various periods of time. The schema consists of three following dimensions: *TIME* ï with the *Time* level, *LOCATION* ï with the *Shops* and *Cities* levels, *COSMETICS* ï with the *Cosmetics* and *Categories* levels.

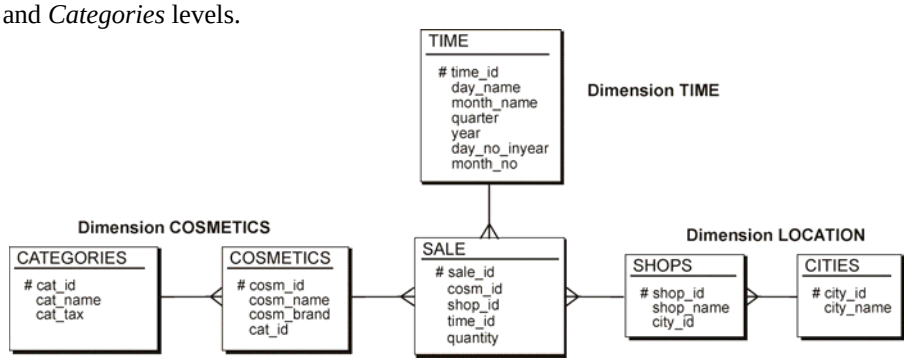


Figure 2. An example DW schema on sale of cosmetics

3. Related Work

The approaches to the management of changes in a DW can be classified into the two following categories that support: (1) schema and data evolution: [5, 11, 12, 15], (2) temporal and versioning extensions [7, 8, 9, 17, 1, 2, 6, 14, 16, 20, 22].

The approaches in the first category support only one DW schema and its instance. When a change is applied to a schema, then all data described by the schema must be converted into a new structure, that incurs high maintenance costs.

In the approaches from the second category, in [7, 8, 9, 17] changes to a DW schema are time stamped in order to create temporal versions. However, [7] and [17] expose their inability to express and process queries that span or compare several temporal versions of data. On the contrary, the model and prototype of a temporal DW presented in [8, 9] supports queries for a particular temporal version of a DW or queries that span several versions. In the latter case, conversion functions must be applied, as data in temporal versions are virtual.

In [14, 16, 20, 22] implicit versions of data are used for avoiding conflicts and mutual locking between OLAP queries and transactions refreshing a DW. As versions are implicitly created and managed by a system, these mechanisms can not be used in the what-if analysis. The same drawback applies to the previously discussed temporal DW that can manage only consecutive versions linearly ordered by time.

On the contrary, [2] proposes permanent user defined versions of views in order to simulate changes in a DW schema. However, the approach supports only simple changes in source tables and it does not deal either with typical multidimensional schemas or evolution of facts or dimensions. Also [6] supports permanent time stamped versions of data. The proposed mechanism, however, uses one central fact table for storing all versions of data. In a consequence, the set of schema changes that may be applied to a DW is limited, and only changes to a dimension schema and a dimension instance structure are supported.

An approach supporting the what-if analysis was presented in [1]. It may be considered as a kind of virtual versioning. A hypothetical query is executed on a virtual structure, called scenario. Then, a system using substitution and query rewriting techniques transforms a hypothetical query into an equivalent query that is run on a real DW. As this technique computes new values of data for every hypothetical query, based on virtual modifications, performance problems will appear for large DWs.

4. Multiversion Data Warehouse: Concept and Operations

4.1. Concept

In our approach, changes made to the structure and content of EDSs are handled in a **multiversion data warehouse** (MVDW) that is composed of the set of its versions (formally defined in [19]). Every version of a MVDW is in turn composed of a schema version and an instance version that represents data described by its schema version. A DW version, called a child version, is derived from a previous DW version, called a parent version. Versions of a DW form a version derivation graph. Each node of this graph represents one version, whereas edges represent **derivedï from** relationships. A version derivation graph is a DAG.

example: changing organizational structure of a company, changing geographical borders of regions, creation and closing shops, changing prices/taxes of products. Real versions are linearly ordered by the time they are valid within.

The purpose of maintaining **alternative versions** is twofold. Firstly, an alternative version is created from a real version in order to support the what-if analysis, i.e. it is used for simulation purposes. Several alternative versions may be created from the same real versions. Secondly, such a version is created in order to simulate changes in the structure of a DW schema. The purpose of such versions is mainly the optimization of a DW structure or system tuning. A DW administrator may create an alternative version that would have a simple star schema instead of an original snowflake schema, and then test the system performance using new data structures.

We distinguish two following groups of operations that modify a DW version:

- operations that change the schema of a DW (further called schema change operations);
- operations that change the structure of a dimension (further called dimension structure change operations).

All the above operations address a particular version of a data warehouse. Applying some of these operations to a given DW version results in the creation of a new DW version, whereas applying other operations may not necessarily cause the creation of a new version. In the latter case, a DW administrator may however explicitly create a new DW version and apply these operations. All operations discussed below can be applied to a real DW version as well as to an alternative version.

4.2. Schema change operations

Schema change operations include the following.

1. **Creating a new level**. The operation creates a level table with a given structure.
   For example, let us assume the creation of level *Regions* with attributes *region_id name* as well as level *CashRegister* with attributes *register_id* and *register_name*. This operation is applied to the same DW version as the operation discussed in point 2.
2. **Connecting a level into a dimension hierarchy**. The operation connects a given level table with its sub- and superlevel tables.
   For example, the *Region* level can be connected as a superlevel of *Cities*. Since this operation does not change the granularity of facts stored in the *Sales* table it may either be executed in the original DW version or in a new one, depending on requirements. Let us also consider inserting level *CashRegister* as a sublevel of *Shops*. In this case the granularity of facts will certainly increase and therefore, the modification is applied to a new DW version. This new version will store sales data for each cash register, whereas old sale data for shops will be stored in a parent DW version.

3. **Disconnecting a level from a dimension**. The operation detaches a given level table from the dimension hierarchy it belonged to.
   For example, the *Product* level could be detached from the *ProductCategory* hierarchy. Since this operation changes the granularity of fact data from finer to coarser, the operation is applied to a new DW version if we do not want to loose historical information about sales of products. In this case, an old DW version will store old data (sales of products), whereas a new one will store new sales data only for each product category.
4. **Removing a level**. The operation removes from the schema previously disconnected level.
   This operation is applied to the same DW version as the operation from point 3.
5. **Adding a new attribute to a level**. The operation augments the structure of a given level table with a new attribute.
   For example, the *City* level could be augmented with attribute *population*. This operation may easily be applied to an existing DW version.
6. **Dropping an attribute from a level**. The operation removes form a level a non primary key or a non foreign key attribute.
   An attribute being removed may be used by analytical queries. In such a case, the queries will no longer be valid. For this operation, a DW administrator decides whether to handle it in a new or an existing DW version.
7. **Changing the domain of a level attribute**. This operation changes the domain of a level attribute.
   If it is applied to a primary key, then all foreign keys that point to this attribute (either in sublevel tables or fact tables) have be modified, that impacts data. This change do not need to be applied to a new DW version as it does not impact user analytical queries. The decision whether to create a new DW version in order handle this change is left to a DW administrator.
8. **Creating a new fact table**. The operation creates a new, empty fact table, without any associations to dimensions.
   Creating a new fact table in a new version of a DW may be helpful for preparing different/alternative data analysis scenarios. The decision whether to create a new DW version in order to handle this change is left to a DW administrator.
9. **Creating a new attribute for a fact table**. The operation creates a new non primary or non foreign key attribute for a given fact table.
   Operation of this type should result in the derivation of a new DW version. Otherwise old fact records would have null values for a new attribute resulting in wrongly interpretable query results. New values of this attribute appear only for records in the new DW version, after loading from external data sources.
10. **Creating an association between a fact table and a dimension**. The operation associates a given fact table with a given dimension.

This operation is implemented by adding a foreign key attribute to a fact table. This operation should be executed in a new DW version for the same reason as in operation 9.

11. **Removing an attribute from a fact table**. The operation removes a non primary key or a non foreign key attribute from a given fact table.
12. **Removing an association between a fact table and a dimension**. The operation is manifested by removing from a fact table an appropriate foreign key.
13. **Removing a fact table**. The operation causes the removal of the whole fact table, previously disconnected from a schema.

Since operations 11, 12, and 13 have a direct impact on existing analytical queries and cause the lost of data, they are applied to a new DW version.

4.3. Dimension instance structure change operators

In addition to the above 13 schema modification operations, we distinguish 5 operations that change the structure of a dimension instance. Since the operations have an impact on results obtained form analytical queries, c.f. [4] they are applied to a new DW version. These operations include:

1. **Inserting a level instance**. The operation inserts a new level instance into a given level. In this case, no action is required on fact data.
   For example, a new shop instance *<1000, 'Marks&Spencer', 20>* can be inserted into the *Shop* level.
2. **Deleting a level instance**. The operation deletes an instance of either a top, or an intermediate, or a bottom level. Appropriate instances of sublevels must also be deleted or reclassified, down to a fact table. In these cases, an old DW version stores fact and dimension data before the change, whereas a new DW version will store new data.
3. **Reclassifying a level instance**. The operation changes the association of a sublevel instance with another superlevel instance.
   For example, product *'Eau de toillette CK'* originally belonging to the category of luxury cosmetics may be move to a new category of popular cosmetics. Old classification (dimension instances) and fact data are stored in an old DW version, whereas new classification and fact data are stored in a new DW version.
   This operation can be useful for the what-if analysis. For example, an alternative version of the example DW could be created in order to analyze the increase or decrease of shops income when taxation of luxury and popular cosmetics differs.
4. **Merging *n* level instances into a new one**. The operation merges several instances of a given level into one instance of the same level.
   For example, *shop_1* and *shop_2* is merged into *shop_3*. This change is reflected in a new DW version. Old fact and dimension data remain in an old version, whereas a new DW version stores only new sales data and dimension data.

5. **Splitting a level instance into *n* new instances**. The operation splits a given instance of a given level into several instances of the same level. For each new instance a split ratio is specified.
   For example, *shop_3* is divided into two shops, namely *shop_31* and *shop_32* at a split ratio equaled to 2/3. This ratio may mean for example that 2/3 of the original budget of *shop_3* is assigned to *shop_31*, and the rest of budget is assigned to *shop_32*. In this case, old fact and dimension data remain in an old version, whereas a new DW version stores only new sales data and dimension data, after shop splitting.

5. Data Sharing

Two consecutive versions of a DW may differ marginally from each other. For example, reclassifying product *'Eau de toillette CK'* from luxury to popular cosmetics requires changes in its dimension instance structure only. Fact data are not impacted by the change. In such a case, the same fact table is common to two DW versions.

A naive approach to dealing with versions of data consists in storing a physical copy of data in every DW version. As the size of data warehouses is of terabytes, this approach is not suitable. Another approach is based on sharing common versions of data between several versions of a DW. On the one hand, data sharing reduces storage overhead, but on the other hand, it introduces query execution time overhead since in order to answer a given query, a system has to infer the set of data belonging to a certain DW version. In a consequence, there exist a trade off between good query execution time and storage overhead.

In our prototype multiversion data warehouse management system (overviewed in [19]) we use data sharing strategy. If however, a user requires very high query performance, DW versions of interest can be explicitly materialized, i.e. each of the DW version will contain its own full set of data. Currently, we are implementing and testing data sharing mechanisms based on bitmaps that describe data sharing between DW versions.

5.1. Bitmap storage

This mechanism is based on storing with every record, in a fact or a level table, the information about all DW versions this record belongs to. At the implementation level, the information about versions a record belongs to is represented as the set of bitmaps, where one bitmap represents one DW version. The number of bitmaps equals to the number of versions sharing data. The number of bits in a bitmap equals to the number of records in a given table. The i^{th} bit in a bitmap, describing version V_m, is set to 1 if the i^{th} record in a table, in DW version V_m, exists in this version. Otherwise the bit is set to 0. This storage is currently implemented in two variants. The first one, called **in-table bitmap storage**, consists in storing every bitmap in a table whose records are shared. The second variant, called **out-of-table bitmap storage**, stores bitmaps out of the table in a separate structure.

In-table bitmap storage. As a simplified example illustrating our in-table bitmap storage let us consider the content of the *Sale* table (from Example 1), as shown below. Initially this table existed in version *R2*. Let us further assume that an alternative version *A2.1* was derived from *R2*. The change in *A2.1* concerned reclassifying *'Eau de toillette CK'* from luxury to popular. In this case, original records from version *R2* are shared also by *A2.1*. To this end, two bitmaps, called *R2* and *A2.1*, are added to the *Sale* table, as shown in Table 1. All sales records are shared by versions *R1* and *A2.1*, i.e. all bits in these bitmaps are set to 1.

Sale (R1)					bitmap	bitmap	bitmap
sale_id	cosm_id	shop_id	time_id	quantity	R2	A2.1	A2.2
1	cosm_1	shop_2	time_1	3	1	1	1
2	cosm_1	shop_2	time_2	1	1	1	1
3	cosm_2	shop_3	time_3	5	1	1	0

Table 1. An example of in-table bitmap storage

Let us further assume that another alternative version *A2.2*, was derived from *R2*. In *A2.2* the *Cosmetics* level instance identified by *cosm_2* was deleted. An appropriate bitmap, called *A2.2*, was added to the *Sales* table, c.f. Table 1. The bitmap informs that sale record identified by *sale_id=3* is shared by versions *R2* and *A2.1* and not by *A2.2*.

Out-of-table bitmap storage. In this storage mechanism, bitmaps are stored out of the table in a separate structure, called **bitmap table**. This table has the following schema: {ROWID, $bitmap_1$, $bitmap_2$, ..., $bitmap_n$}, where ROWID is a physical address of a record, whereas $bitmap_1$, $bitmap_2$, ..., $bitmap_n$ are bitmaps, each of which represents a DW version and its records. A *bitmap table* is created for every DW table, either fact or dimension, whose data are shared between DW versions. As an example let us consider table *Sale* and its content as shown in Table 1. The out-of-table storage for the *Sale* table is shown in Table 2. The *ROWID* values in the *Sale bitmap table* correspond to the *ROWID* values in the *Sale (R1)* table. For example, record located at 0x0009 address is shared by versions *R2*, *A2.1*, and *A2.2*.

Sale (R1)					
ROWID	sale_id	cosm_id	shop_id	time_id	quantity
0x0009	1	cosm_1	shop_2	time_1	3
0x000A	2	cosm_1	shop_2	time_2	1
0x000C	3	cosm_2	shop_3	time_3	5

Sale bitmap table			
ROWID	R2	A2.1	A2.2
0x0009	1	1	1
0x000A	1	1	1
0x000C	1	1	0

Table 2. An example of out-of-table bitmap storage

The in-table bitmap storage decreases concurrent processing since locking a record also causes locking version information. Moreover, reading version information requires reading whole records, thus more data is transferred from disk. The opposite characteristics has the out-of-table storage. Locking a record does not require locking a bitmap and vice versa. Reading version information is accessed in an appropriate *bitmap table* which is much smaller than fact tables. Additionally, a table being shared and its bitmap table can be placed on different disks allowing parallel access. However, the out-of-table bitmap storage requires joining a table with data with a bitmap table, that introduces additional query processing overhead.

5.2. Experimental results

In this paper we focus on evaluating our in-table as well as out-of-table bitmap storage. Our test environment is characterized as follows:

- a PC with Celeron 333MHz, 512MB of RAM, Linux;
- an Oracle 9.2 database with 73MB SGA;
- total size of a fact table, being shared by several DW versions equaled to 790 MB (11 million of records);
- queries being tested were based on a standard TPC-D benchmark;
- queries were run on a standard star schema with the *Sales* fact table as well as *Products*, *Times*, *Shops*, and *Customers* as dimension tables.

In order to test query response time overhead we run various TPC-D benchmark queries; five of them are discussed below. The queries address one particular DW version, out of 12 versions stored in our MVDW.

Q1:
```
select sum(quantity)
from   sales
where  time_id >= '00/03/01' and time_id <= '00/04/30'
and    quantity > 5;
```

Q2:
```
select product_name, sum(quantity)
from   sales, products
where  sales.product_id = products.product_id
and    category = 'CAT_02'
group  by product_name;
```

Q3:
```
select city, sum(quantity)
from   sales s, shops sh, products p
where  s.product_id = p.product_id
and    s.shop_id = sh.shop_id
and    p.category in ('CAT_01','CAT_02')
group  by city;
```

Q4:
```
select sh.city, sum(quantity)
from   sales s, shops sh, products p, customers c
where  s.product_id = p.product_id
and    s.shop_id = sh.shop_id
and    s.customer_id = c.customer_id
and    p.category in ('CAT_01','CAT_02')
and    c.city in ('CITY_01','CITY_08')
group  by sh.city;
```

Q5:
```
select month, sum(quantity)
from   sales s, shops sh, products p, customers c, times t
where  s.product_id = p.product_id
and    s.shop_id = s.shop_id
and    s.customer_id = c.customer_id
and    s.time_id = t.time_id
and    p.category in ('CAT_01','CAT_02')
and    c.city in ('CITY_01','CITY_08')
and    sh.city in ('CITY_02','CITY_03')
group  by month;
```

The queries were run for three different cases of data sharing between DW versions. In the first case (marked in Figure 3 as *mat. version*) every DW version was materialized, i.e. it stored its full set of data, without any data sharing. In the second case (marked in Figure 3 as *in-table stor.*) the in-table-storage was

implemented for data sharing. Finally, in the third case (marked as *out-of-table stor.*) the out-of-table storage was implemented for data sharing.

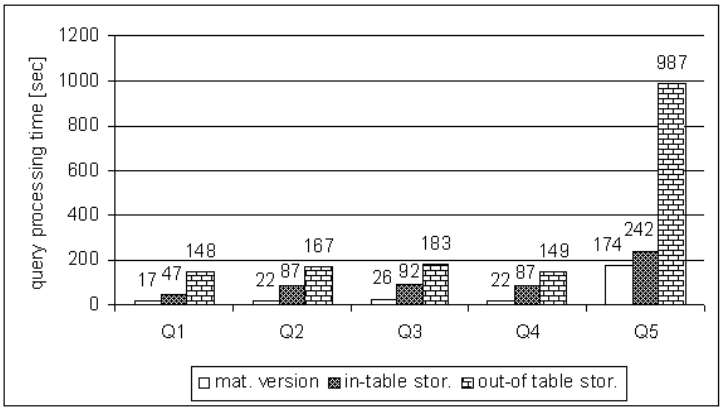


Figure 3. Experimental results of processing queries on multiversion DW instance

As we can observe from Figure 3, the in-table storage introduces smaller time overhead than the out-of-table storage. The in-table-storage, in the best case, slows down query processing by 1.4 (query Q5) as compared to processing of the same query on a fully materialized version. (cf. processing time of Q5 for in-table storage divided by processing time of Q5 for out-of-table storage). In the worst case, the in-table-storage caused 4 times increase in query processing time as compared to a fully materialized DW version (cf. query Q2 and Q4). The out-of-table storage gives worse results as it slows down query processing from 5.7 (Q5) to 8.7 (Q1) times as compared to the query processing in a fully materialized DW version. As our experiments show, the time overhead introduced by data sharing has less impact on total query processing time for more complex queries.

5.3. Additional storage overhead

The management of a multiversion DW requires additional data storage overhead for metadata information. The information concerns: (1) multiversion schema and (2) multiversion data.

The metadata information about a DW schema is stored in a metaschema (cf. [3]), implemented as 17 tables. The average length of a metadata record equals to 120B. Therefore, the average amount of metadata information describing one version of a simple snowflake schema ranges from 10kB to 20kB. For example, a DW schema from Figure 2 requires 11kB metadata information. Thus, metadata information about a MVDW schema is very small.

A more substantial storage overhead is caused by data sharing information, as discussed in Section 5.1. If a bitmap is implemented as a separate attribute in a table (the in-table storage), then the smallest attribute size is 1B. In this case, the required additional storage is computed by the following formula: `nb_of_bitmaps x 1B x nb_of_records`. If however, every bitmap is stored as a separate bit, then the storage will be much smaller. The out-of-table storage requires additional space for storing

physical addresses of records. For example, in an experimental DW that we used, data sharing information for 12 DW versions took 126MB and 315MB of disk space for the in-table and out-of-table storage, respectively.

7. Summary, Conclusions, and Future Work

Handling changes in external data sources, supplying data to a DW, and applying the changes to the DW become important research and technological issues. Structural changes to a DW schema applied inappropriately may result in wrong analytical results. Most of commercial DW systems existing on the market have static structure of their schemas and relationships between data. In a consequence, they are not well suited for handling of any changes that occur in a real world. Moreover, the existing systems do no support the creation of alternative business scenarios for the purposes of the what-if analysis. Research prototypes and solutions to this problem are mainly based on temporal extensions that limit their use. Our approach to this problem is based on a multiversion data warehouse.

In this paper we briefly presented our concept of a MVDW, we discussed types of versions needed in such a warehouse, overviewed schema change operators and dimension instance structure change operators. A unique feature of our model of a MVDW is its ability to represent *alternative versions* of a DW (required for the what-if analysis) as well as physical separation of different DW versions, unlike in other approaches.

Since two DW versions may differ marginally, we proposed data sharing mechanism between these DW versions. The mechanism uses bitmaps describing data sharing between DW versions. We evaluated experimentally two alternative implementations, namely the in-table as well as out-of-table bitmap storage. The obtained results, presented in this paper, show that the in-table-storage slows down query processing by 1.4 in the best case, that still can be an acceptable delay. The presented concept of a MVDW and data sharing techniques are being developed an implemented in a project whose goal is to build a multiversion data warehouse system.

Future work will concentrate on: (1) experimental evaluation of scalability of our storage technique with respect to the number of versions as well as volumes of data being shared, (2) developing new mechanisms of indexing multiversion data, (3) developing a model of transactions for a MVDW, and (4) analyzing inter-version and intra-version integrity constraints.

References

[1] Balmin, A., Papadimitriou, T., Papakonstanitnou, Y.: Hypothetical Queries in an OLAP Environment. Proc. of the VLDB Conf., Egypt, 2000

[2] Bellahsene, Z.: View Adaptation in Data Warehousing Systems. Proc. of the DEXA Conf., 1998

[3] Bňbel B., Kr· likowski Z., Morzy T., Wrembel R.: Tranasction Concepts for Supporting Changes in Data Warehouses. Proc. of the 6th International Conference on Enterprise Information Systems (ICEIS), Porto, Portugal, April, 2004 (to appear)

[4] Bňbel B., Morzy T., Wrembel R.: Querying a Multiversion Data Warehouse. Proc. of the SOFSEM Conference, Czech Republic, 2004
[5] Blaschka, M. Sapia, C., Hº fling, G.: On Schema Evolution in Multidimensional Databases. Proc. of the DaWak99 Conference, Italy, 1999
[6] Body, M., Miquel, M., B®dard, Y., Tchounikine A.: A Multidimensional and Multiversion Structure for OLAP Applications. Proc. of the DOLAP'2002 Conf., USA, 2002
[7] Chamoni, P., Stock, S.: Temporal Structures in Data Warehousing. Proc. of the Data Warehousing and Knowledge Discovery DaWaK, Italy, 1999
[8] Eder, J., Koncilia, C.: Changes of Dimension Data in Temporal Data Warehouses. Proc. of the DaWak 2001 Conference, Germany, 2001
[9] Eder, J., Koncilia, C., Morzy, T.: The COMET Metamodel for Temporal Data Warehouses. Proc. of the 14th Int. Conference on Advanced Information Systems Engineering (CAISE'02), Canada, 2002
[10] Gyssens M., Lakshmanan L.V.S.: A Foundation for Multi-Dimensional Databases. Proc. of the 23rd VLDB Conf., Grece 1997
[11] Hurtado, C.A., Mendelzon, A.O.: Vaisman, A.A.: Maintaining Data Cubes under Dimension Updates. Proc. of the ICDE Conference, Australia, 1999
[12] Hurtado, C.A., Mendelzon, A.O.: Vaisman, A.A.: Updating OLAP Dimensions. Proc. of the DOLAP Conference, 1999
[13] Jarke, M., Lenzerini, M., Vassiliou, Y., Vassiliadis, P.: Fundamentals of Data Warehouses. Springer-Verlag, 2000, ISBN 3-540-65365-1
[14] Kang, H.G., Chung, C.W.: Exploiting Versions for Onï line Data Warehouse Maintenance in MOLAP Servers. Proc. of the VLDB Conference, China, 2002
[15] Koeller, A., Rundensteiner, E.A., Hachem, N.: Integrating the Rewriting and Ranking Phases of View Synchronization. Proc. of the Int. Workshop on Data Warehousing and OLAP (DOLAP), USA, 1998
[16] Kulkarni, S., Mohania, M.: Concurrent Maintenance of Views Using Multiple Versions. Proc. of the Intern. Database Engineering and Application Symposium, 1999
[17] Mendelzon, A.O., Vaisman, A.A.: Temporal Queries in OLAP. Proc. of the VLDB Conference, Egypt, 2000
[18] Letz C., Henn E.T., Vossen G.: Consistency in Data Warehouse Dimensions. Proc. of the Intern. Database Engineering and Applications Symposium (IDEAS'02), 2002
[19] Morzy, T., Wrembel, R.: Modeling a Multiversion Data Warehouse: A Formal Approach. Proc. of the Int. Conf. on Enterprise Information Systems - ICESI'2003, France, 2003
[20] Quass, D., Widom, J.: Onï Line Warehouse View Maintenance. Proc. of the SIGMOD Conference, 1997
[21] Roddick J.: A Survey of Schema Versioning Issues for Database Systems. In Information and Software Technology, volume 37(7):383-393, 1996
[22] Rundensteiner E., Koeller A., and Zhang X.: Maintaining Data Warehouses over Changing Information Sources. Communications of the ACM, vol. 43, No. 6, 2000
[23] Vaisman A.A., Mendelzon A.O., Ruaro W., Cymerman S.G.: Supporting Dimension Updates in an OLAP Server. Proc. of the CAiSE'02 Conference, Canada, 2002

SCIENTIFIC PAPERS UNIVERSITY OF LATVIA. 2004. VOLUME 672, Datbases and Information Systems
232–245 P.

From Temporal Data Mining and Web Mining To Temporal Web Mining

Mireille Samia * Stefan Conrad

Institute of Computer Science
Databases and Information Systems
Heinrich-Heine-University Düsseldorf
D-40225 Düsseldorf, Germany
{samia|conrad}@cs.uni-duesseldorf.de

Abstract. Treating data with temporal information as temporal data, and not as static, is becoming the concern of different domains. Temporal Web mining is a new approach, which deals with real time data. Temporal Web mining extends temporal data mining and Web mining. It concerns the Web mining of data with significant temporal information. It can be used in different areas, such as finance, engineering, environmental sciences, earth sciences, and medicine. Its main goal is to analyze local and Web data in real time in order to discover relevant temporal information. This paper presents the new concept Temporal Web Mining (TWM), focuses on its architecture, and discusses some of its application scenarios, such as volcanism and seismology, and flooding.

Keywords. Temporal Web Mining (TWM), TWM Architecture

1 Introduction

The ability to obtain real time data from multiple different sources, to share it, and to analyze it is of great importance for the development of the information society. Temporal Web mining aims at treating data with temporal information in real time over the Web. It extends temporal data mining and Web mining. Its main goal is to query local and Web data in real time, analyze these temporal sequences in order to discover previously unknown important temporal knowledge. Another purpose is to introduce prediction as a main issue in Web mining, specifically Web content mining. Furthermore, it uses Web data with temporal information in the temporal data mining process. Temporal Web mining can be used in different domains, such as finance, engineering, environmental sciences, medicine, and earth sciences. One of its application scenarios is flooding. Flooding can threaten the public safety and potentially damage properties [14]. Combining and analyzing data about flooding, such as historical flood damage records and flooding data hazards, flooding relationships to other

* **This work is supported by the German Academic Exchange Service (Deutscher Akademischer Austausch Dienst DAAD).**

hazards (such as earthquakes, landslides) as well as auxiliary information (such as meteorological data), in real time locally and/or over the Web, can enhance the capability of predicting a flooding.
Another instance is volcanoes and earthquakes. Data about volcanic and seismic phenomena, such as historical volcanic data, historical seismic data, data about any current volcanic or seismic activity, are received from different sources (such as stations), and combined with other data, such as meteorological data, in order to be queried in real time, locally and/or over the Web. Hence, the obtained data is analyzed in order to extract the needed temporal data which can be a way to forecast to a certain extent earthquakes and volcanic activities, and a possibility to enhance the safety measures.
Another goal of temporal Web mining is to make good use of the vast amount of data that has been accumulated over the past decades. It encourages and aims at contributing in building a common reference data warehouse which improves data record, data exchange, data manipulation, data standardization and data analysis between different institutions and communities, such as researchers.

In this paper, section 2 gives a brief overview of temporal data mining and Web mining, and discusses related work. In section 3, we define temporal Web mining after describing its process. In section 4, we present the temporal Web mining architecture, and analyze it. Then, we discuss some of its application scenarios in flooding, and earth sciences. Section 5 concludes this paper and points out our directions for future work.

2 Overview and Related Work

This section gives a brief overview of temporal data mining and Web mining before discussing further related work.

2.1 Overview

Temporal Data Mining

Data mining is the process of discovering and extracting relevant hidden relationships and previously unknown interesting patterns that exist in information repositories, such as databases or data warehouses [8].

Temporal data mining mines data collected over time. It concerns the analysis of temporal data and the search for important hidden relationships and patterns between data with temporal information. In other words, its goals are to discover and analyze previously unknown hidden relations between temporal data, and to predict temporal sequences. Depending on the type of temporal data, the approaches to solve the mining problem could differ [1]. A temporal sequence is defined as a sequence of nominal symbols from a particular alphabet, and a time series as a sequence composed of continuous real valued elements.

The discovery of relations between temporal data starts with the representation and modeling of the temporal data. The data can be represented into time series data in

either continuous or discrete, linear or non-linear models, stationary or nonstationary models and distribution models (e.g., time-domain representation or time series model representation) [12]. It can also be represented into time series data using a continuous or a discrete transformation (e.g., discrete fourier transform, discrete wavelet transform and discretization transformation). To find subsequences of events, a sliding window of size w can be used, and placed at every possible position of the sequence [1].
After representing the temporal data, similarity measures between sequences of events are defined. For example, the distance between temporal characteristics in a continuous time domain can be measured using the Euclidean squared distance function. Measuring similarities in discrete spaces can be done using the Shape Definition Language [1].
After finding similarities between sequences of events, temporal data mining techniques are applied, such as the discovery of association rules, classification, and clustering [1, 12].

Web Mining

The Web is a large distributed repository of data, which provides the opportunity to use data mining techniques to analyze Web data. The application of data mining techniques to the Web data is called Web mining.

Web mining is defined as the application of data mining techniques to automatically discover and extract useful information within the Web data and services [6].

Web mining is divided into the following subtasks [6, 10]:

1. Resource Finding: Intended Web documents are retrieved.
2. Information Selection and Pre-processing: Specific information is automatically selected and pre-processed from the retrieved Web resources.
3. Generalization: At individual Web sites and across multiple sites, general patterns are automatically discovered.
4. Analysis: The mined patterns are analyzed.

Web mining can be classified into three major categories [10]:

– *Web Content Mining*: Web document content patterns are automatically discovered.
– *Web Usage Mining* or *Web Log Mining*: Web server access patterns are automatically found.
– *Web Structure Mining*: Hypertext and linking structure patterns are automatically discovered.

2.2 Further Related Work

Roddick and Spiliopoulou [19] discuss two directions of temporal data mining. One concerns the discovery of causal relationships among temporal events that may be causally related. The other concerns the identification of similar patterns within the same time sequence or among different time sequences. Following Antunes and

Oliveira [1], the data should be viewed as a sequence of events, and not as an unordered collection of events. Lin et al. [12] discuss two kinds of fundamental problems in temporal data mining. One is the similarity search problem which is defined as the problem of seeking for sequences that are similiar to a given query sequence. The other is the periodical problem which is the problem of finding periodic patterns or cyclicity occurring in time-related databases.

The World Wide Web is growing rapidly. Its large amount of data is widely distributed and lacks a unifying structure. Web mining assists different researches in intelligently searching and extracting relevant data. Craven et al. and Madria et al. [5, 13] focus on Web mining in order to discover new knowledge from the information available on the Web. Madria et al. [13] suggest a Warehouse of Web data, called Whoweda, containing Web data to support its research issues of Web mining. Following Leung et al. [11], Web Archeology's approach is to explore the content of the Web and locate relevant information sources. It studies pages collected from millions of Web sites. However, the lack of structure limits the discovery of important information and the data retrieval performance.
Following Kosala et al. [10], the problem of personalization of Web information is often associated with the type and presentation of data. For example, although the data can contain temporal knowledge, it is treated as static.

The creation of Web archives is receiving a substantial interest. The Internet Archive project [3, 17] aims at building an Internet library with the purpose of preserving Web information for a long-term and of offering free access to historical digital collections. It receives its data archive from third parties. The main contributor is Alexa Internet which is a Web navigation service [4]. The Austrian On-Line Archive (AOLA) [2, 17] is a project which creates a permanent archive documenting the evolution of the Austrian Web space. Its aim is to build an archive of the national Austrian Web space. In the Austrian On-Line Archive Processing (AOLAP) project [18], Web data obtained from crawls of the Austrian Web space as part of AOLA and information collected from additional sources, such as the WHOIS database, are combined. The resulting data warehouse facilitates data analysis and the exploration of the selected Web pages.

Temporal Web Mining (TWM) is an extension of temporal data mining and Web mining. We propose to combine temporal data mining and Web mining in order to deal with real time data and multiple sequences. Real time data received by one or multiple sources is analyzed using temporal data mining techniques in order to extract new temporal information. The new temporal information is sent over the Internet and is combined with auxiliary data from other sources. Then, it is used in the Web mining process in order to discover new relevant temporal data in real time and to predict previously unknown temporal information. TWM supports the temporal aspect of Web data by mining Web data with temporal information as temporal data, and not as static data. Its purpose is to introduce prediction as a main issue in Web mining, specifically Web content mining. In other words, TWM aims at predicting temporal data from the content of the Web data. Furthermore, TWM uses Web data, such as temporal data from the Web, in the temporal data mining process.

The Web seems to be too huge and lacks of structure for effective data mining and data warehousing. For this reason, our goal is to encourage building a common reference data warehouse. The essential difference between the existing data warehouses (such as [3, 13]) and our common reference data warehouse is that the analysis of the data derived from the application of temporal data mining techniques and Web mining techniques is stored in addition to the data received from different sources, such as conventional data warehouses, and the World Wide Web. This can enhance the communication, coordination and integration of different societies and communities, such as researchers. It improves data sharing, data exchange and data manipulation between these different communities and societies. Since the analysis of the data obtained by applying the previous mentioned mining techniques is also saved in our data warehouse, TWM improves different mining techniques. New previously unknown crucial relationships and patterns can be extracted. This contributes to discover, for instance, standard patterns and relationships that can be used in different mining techniques, such as classification. It can also be a clue for representing data, defining similarities between sequences, and finding periodic patterns. Interpreting data in real time and improving data analysis enhance data prediction.

3 Temporal Web Mining Definition

In this section, we define temporal Web mining after describing its process.

3.1 Description of the Temporal Web Mining Process

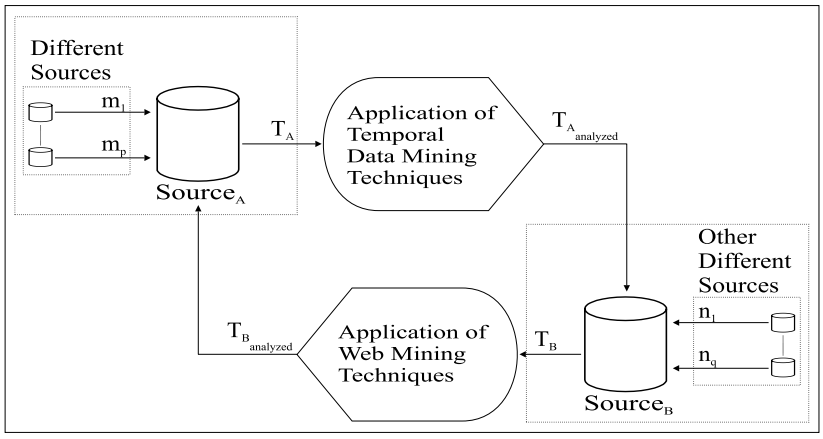


Fig. 1. An Overview of the Temporal Web Mining Process

In Figure 1, $Source_A$ receives data with temporal information ($m_1..m_p$) from one or different sources in real time. The combined data T_A is analyzed using temporal data

mining techniques. This analyzed temporal data $T_{A_{analyzed}}$ is forwarded to $Source_B$ in real time over the Internet, and added to data obtained from other sources ($n_1..n_q$). This temporal data T_B is analyzed using Web mining techniques. The derived previously unknown temporal data $T_{B_{analyzed}}$ can reveal new valuable information which can support the data analysis of $Source_A$. In this case, $T_{B_{analyzed}}$ is forwarded to $Source_A$, and is used in the data analysis of this source by the application of temporal data mining techniques.

In the TWM process, data with temporal information from the Web is considered as temporal data, and not as static. This temporal data is used in the Web mining process in order to predict new temporal knowledge from the content of the Web data. Then, TWM extends Web mining. TWM aims at using data with temporal information from the Web as temporal data, and at introducing prediction as a main aspect in Web mining, specifically Web content mining. Moreover, by sending the analysis of $Source_B$ to $Source_A$, TWM uses data from the Web, such as temporal Web data, in the temporal data mining process.

3.2 Definition of Temporal Web Mining

Temporal data discovered by the application of temporal data mining techniques is used in the Web mining process in order to retrieve useful data with temporal information in real time over the Web. The derived useful data with temporal information discovered by the application of Web mining techniques is used again in the temporal data mining process.

We define Temporal Web Mining (TWM) as the process of discovering, extracting, analyzing, and predicting data with significant temporal information from the temporal information discovered by the application of temporal data mining techniques, and applying Web mining techniques to this data in real time over the Web (cf. Figure 1)[21].

4 Temporal Web Mining Architecture

In this section, we present the architecture of temporal Web mining, and analyze it. Then, we discuss some of its application scenarios in volcanism and seismology, and in flooding.

4.1 Description of TWM Architecture

The architecture of temporal Web mining consists of three main components: original source component, destination source component and global source component (cf. Figure 2).

The aims of our new architecture are to discover, extract, query, analyze, exchange, record, use, and predict (if possible) data between these main components in real time and over the Web.

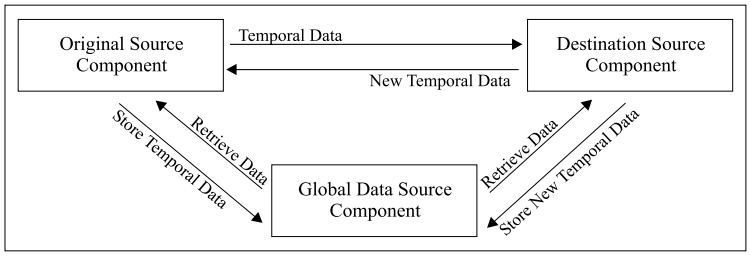


Fig. 2. Main Components of TWM Architecture

Figure 3 represents the architecture of TWM. It describes the three TWM main components. The connections between these components depict the exchange of data in the indicated directions. *Temporal data* represents the data derived from the application of temporal data mining techniques. *New temporal data* is the data obtained from the application of Web mining techniques.

Original Source Component

In Figure 3, *multiple different sources* interact, in real time, with each other to retrieve, combine, and analyze data.

One or more of these *multiple different sources* receive real time data. The obtained temporal data is collected in order to be selected and explored. Applying temporal data mining techniques, this temporal data is analyzed. From the *temporal data* database, it is sent to the global source component, where it is saved. Furthermore, in order to reveal more important information, this temporal data may be combined with other data retrieved from the global source data component. It can also be added to other data received from the destination source component. In this case, it is sent over the Internet to the destination source component.

Destination Source Component

The destination source component provides an environment for decision support using mining techniques. The *new temporal data* database receives the analyzed temporal data from the original source component in real time and over the Web (cf. Figure 3). It is examined and, if needed, combined with auxiliary data acquired from *other multiple different sources* or from the global source component. Then, it is analyzed in order to extract derived temporal data. The new temporal data is sent back, in real time over the Internet, from the *new temporal data* database to one or more sources of the original source component. It is also saved in the global source component.

Global Data Source Component

The global source component consists of the *global data source database* (also called common reference data warehouse) and the *global data source backup* database. The *global data source database* receives data from the original source component and the

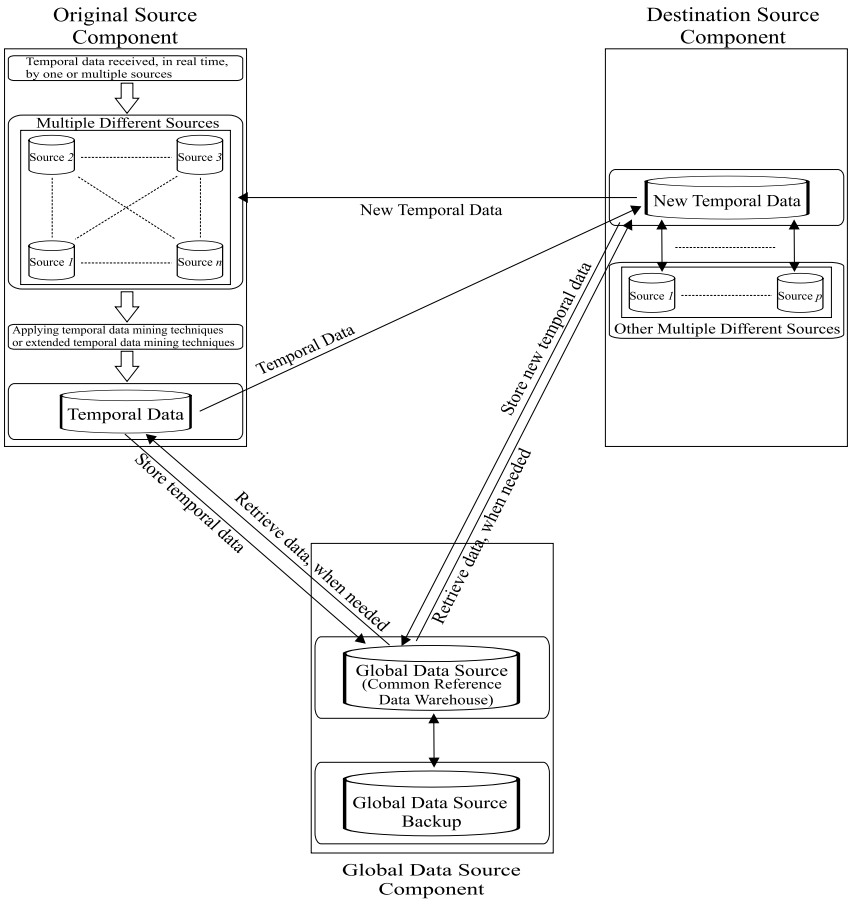


Fig. 3. Temporal Web Mining Architecture

destination source component. It backups this data in the *global data source backup* database.

Whenever needed, the *global data source component* forwards the requested data to the *original source component* or to the *destination source component*. Copies of the same data are sometimes requested similtaneously from different sites. Instead of sending that data to just one site at a given time, it can be sent to more than one site at more or less the same time by using our multicast network (cf. Figure 4) [20]. Hence, our multicast network is responsible for handling concurrent data requests of multiple sites. It is provided with additional functions, such as backup and error recovery. It consists of a source node connected to a number of networks. Each of these networks is connected to an intermediate site (i.e. a router). Each intermediate site is connected to several networks which lead to a number of destination sites. The source node is

the *global data source component* and the destination sites are those that requested concurrently the same data.

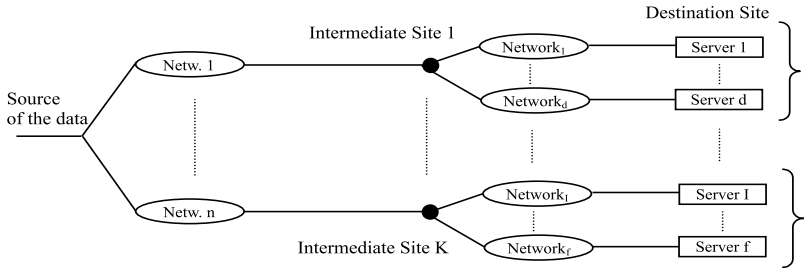


Fig. 4. Multicast Network

4.2 Analysis of the TWM Architecture

TWM collects, exchanges, interprets, and predicts (if possible) data between different sources, in real time and over the Web.

Causal relationships among temporal related events have to be discovered by using mining techniques, in order to find the cause of an event and predict its result, if necessary.

The quality of data is one of the most crucial factors to be taken into account [9, 16]. Data is collected from different sources in real time in order to be queried and analyzed. Then, relevant data is to be distinguished from the noise, or any data error. In other words, the ability to deal with any abnormality in data is to be taken into account in order to be reduced.

Since temporal Web mining deals with data changing rapidly over the time, delay and interruptions in data transfer should be minimized. This can be done by optimizing the query execution, by finding a correct representation of the temporal knowledge in question, and by selecting the right path to retrieve the required data.

Because it is difficult to locate and track appropriate data from the huge number of semi-structured documents, our aim is to contribute in building logical structured information sources, such as a common reference data warehouse, that combine data from multiple different sources. This minimizes the cost of data access time, as well as delay, and reduces data redundancy. The structure can be optimized by analyzing the data, the Web logs and the user's behavior. For instance, the user's behavior can be studied by examining Web data, such as the page requested, the time and frequency of request of pages. In other words, Web mining techniques can be applied to interpret and predict user's behavior.

Having a common reference data warehouse, temporal Web mining improves data sharing, data exchange and data manipulation between different societies, institutions

and communities. In Figure 3, the *global data source component* establishes a data exchange system that may be used by different communities, such as research communities and different institutions, in order to record, exchange, use, process, and analyze data, assembled from different sources, such as the World Wide Web or conventional data warehouses. It also acts as a backup of the data exchange communities, because the data is regularly collected and stored in the *global data source component* from databases worldwide. It is a possibility to unify the knowledge all over the world for the rapid advance of many fields.

The *global data source component* sends requested data to one or more sites, when needed. Multiple different sites may make concurrent requests of the same data. Instead of delivering this data to just one of these sites at a given time, it is sent to the sites in question at more or less the same time. Our multicast network supports concurrent data requests of multiple sites. In Figure 4, whenever data is needed to be multicasted to different destinations, data is forwarded from the *source of the data* (i.e. *global data source component*) along a distribution tree to each receiver (i.e. *destination site*)[20]. The destination sites (such as *original source component* or *destination source component*) do not communicate directly with the *global data source component* (cf. Figure 3). However, they communicate with their corresponding intermediate site. Each intermediate site supports backup and error recovery functions [20]. Consequently, whenever an intermediate site receives a corrupted piece of data or does not receive the requested piece of data, a message is sent back to the *source of data* requesting the retransmission of the needed data. Moreover, if a destination site does not receive the correct data, a message is sent back to the *intermediate site* asking it to retransmit the corresponding data. Then, a *destination site* does not communicate directly with the *global data source component*. The load on the *global data source component* is reduced which helps to reduce its response time. More clearly, the workload is distributed among the *global data source component* and the *intermediate sites*. This helps to minimize delay and interruptions in data transfer, as well as data loss and data errors.

In Figure 3, our *common reference data warehouse* creates a data exchange standard by, for instance, finding a standard representation for shared data, and by creating common patterns from this data. This can also minimize the cost of data access time, and reduce redundancy. For instance, the *global data source component* gives the possibility for seismologists to use the global seismological data and its analysis, collected in the *global data source component* from many different institutions worldwide, in their data analysis. This makes processing earthquake data, in real time over the Web, easier, faster and more accurate, and helps to extract and predict more relevant important knowledge.

4.3 Application Scenarios for TWM

Collecting and exchanging data between different sources, in real time and over the Web, provide an opportunity for various societies and communities to share information instantly and analyze it in order to contribute to comprehension of different issues that have far-reaching implications for the humankind and environment.

In TWM, different application scenarios can be applied, such as:

- Volcanoes and Earthquakes

  Volcanoes have played an important role in forming and modifying the Earth. More than 80% of the Earth's surface above and below the sea level is of volcanic origin [7]. The keeping of a detailed history of the changes in a volcano and its surroundings helps to characterize the behavior of the corresponding volcano. Visible changes are any observable and often measurable features by researchers that might reflect a change in the state of a volcano. For instance, scientists collect the eruptive products and gases for laboratory analysis, make temperature measurements of lava and gas, and so on. Invisible changes are any changes, which are not visible to the human eye, but measurable by precise and sophisticated instruments and sensors, such as variations in gas compositions.

  An earthquake generates seismic waves which can be sensed and recorded over a wide range of frequencies and seismic amplitudes. The severity of an earthquake depends on many factors, such as the measure of the amplitude of the seismic waves (i.e. the magnitude of the earthquake), the building design and other structures, the geologic conditions, the density of the population and constructions in the area affected by the earthquake.

  Volcanism and seismic activities are often closely related, responding to the same dynamic Earth forces. Real time volcanic data and seismic data can be obtained from, for instance, sensing devices located at different stations. Combining, in real time, from different sources over the Web, the recording and analysis of volcanic phenomena and seismograms from many earthquakes helps to better understand and analyze the Earth's deep interior.

  Historical volcanic data, historical seismic data, and historical meteorological data are examples of historical data. Instances of any current data related to a volcanic event are eruption date and time, volcano location, volcano frequency of occurrence, volcano magnitude, temperature of the lava, seismic data, and meteorological data. Historical data and any current data related to a volcanic event can be in real time exchanged and analyzed between different institutions over the Web in order to better understand, determine and analyze the relations between volcanism and seismic activities. More specifically, volcano data can be date and time of eruption, magnitude, volcano surface temperature, variations in gas compositions, historical volcanic eruption, type of volcanic eruption, volcano elevation, and frequency of occurrence. Seismic data can be magnitude, location, starting and ending date and time of the earthquake, frequency of occurrence, temperature measurements, and barometric pressure. In Figure 3, whenever a volcanic activity or a seismic activity occurs, volcano data and seismic data are analyzed instantly, if required with other data (such as wind speed data, wind direction data) from *multiple different sources* (such as a station where a wind sensor is located, a seimic station) or from the *global data source*. The obtained temporal data is sent, in real time over the Internet, from the *temporal data* database to the *new temporal data* database, and is also saved in the *global data source*. In the *new temporal data* database, it is combined, if necessary, with new information (such as meteorological data, historical seismic data) from

the *new temporal data* database, from *other multiple different sources* (such as meteorological station, seismic station) or from the *global data source*. Then, it is analyzed in order to extract new temporal data that can reveal crucial information about volcanic or seismic activities. The new derived temporal data is sent back to the original source and saved in the *global data source*. Analyzing this data connects different stations and provides mutual support to increase the possibility of searching for any warning sign that can forecast future volcanic or seismic activities.

In this application scenario, temporal Web mining helps to create a better image of the Earth's deep interior, figure out any new idea about volcanoes or earthquakes, and improve the capability for predicting future volcanic or seismic activities. Furthermore, precise determination of every active or inactive volcano location, the detailed analysis of their criteria, as well as the study of damages caused by volcanoes and earthquakes reduce their losses and hazards by enhancing the safety measures and emergencies. They also help researchers, architects, structural engineers, or those who are working in the building or construction domain in their researches and the structure's building and design [15].

- Flooding

  The assembling of flood data and flood causes data between different stations, in real time and over the Internet, creates a more complete documentation and understanding of historical flood data (such as flood events, flood causes, flood damages). Hence, flood data can be better analyzed, which helps to enhance the capability of forecasting a flooding, to warn about a flooding, and to present an effective disaster planning to reduce flood losses and hazards and improve the public safety measures, as well as the emergency management.

5 Conclusion and Outlook

Temporal Web Mining (TWM) is a new concept which combines and extends temporal data mining and Web mining. Its purpose is to introduce prediction as a main issue in Web mining, specifically Web content mining, and to use Web data, such as temporal data from the Web, in the temporal data mining process. It can be used in different domains, such as finance, engineering, medicine, environmental sciences, and earth sciences. Its main goal is to deal with temporal data in real time over the Web, in order to discover, in real time, patterns and relationships among time ordered events that may be causally related. Consequently, some disasters, such as an earthquake, an eruption of a volcano, or a flooding can be to a certain extent forecasted, and effective disaster plannings can be enhanced. It also encourages and aims at providing assistance in building a common reference data warehouse which gives better opportunities for data record, data sharing, data manipulation, data standardization and data analysis between different societies, communities, and institutions.

In this paper, we give a brief overview of temporal data mining and Web mining, and

discuss related work. We describe the temporal Web mining process, and define temporal Web mining. Then, we present the architecture of TWM, analyze it, and apply application scenarios in volcanism and seismology, and flooding.

Future directions to our research include the improvement of the quality and delivery of information, as well as data analysis, by finding an appropriate data representation. Analyzing the users' behavior in the Web can help to determine an effective query execution plan. Estimating the cost of a query execution and reducing the cost of selecting the adequate path to retrieve the required data can improve the data extraction and data analysis capability. Another work can be to develop an accurate system for data analysis which preserves security and prohibits the access to unneeded confidential information in data records.

References

1. Claudia Antunes and Arlindo Oliveira. Temporal Data Mining: an Overview. In *KDD Workshop on Temporal Data Mining*, pages 1–13, San Francisco, 2001.
2. Austrian On-Line Archive. Website. http://www.ifs.tuwien.ac.at/∼aola.
3. The Internet Archive. Website. http://www.archive.org.
4. Alexa: The Web Information Company. Website. http://www.alexa.com.
5. Mark Craven, Dan DiPasquo, Dayne Freitag, Andrew K. McCallum, Tom M. Mitchell, Kamal Nigam, and Seán Slattery. Learning to Extract Symbolic Knowledge from the World Wide Web. In *Proceedings of AAAI-98, 15th Conference of the American Association for Artificial Intelligence*, pages 509–516, Madison, US, 1998. AAAI Press, Menlo Park, US.
6. Oren Etzioni. The World-Wide Web: Quagmire or Gold Mine? *Communications of the ACM*, 39(11):65–68, 1996.
7. The Federal Emergency Management Agency (FEMA). (2003). Hazards: Volcanoes. Retrieved October 13, 2003, from http://www.fema.gov/hazards/volcanoes/volcano.shtm.
8. William J. Frawley and Gregory Piatetsky-Shapiro. Knowledge Discovery in Databases. AAAI Press/The MIT Press, 1991.
9. Jochen Hipp, Ulrich Güntzer, and Udo Grimmer. Data Quality Mining - Making a Virtue of Necessity. In *Proceedings of the 6th ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery (DMKD 2001)*, pages 52–57, Santa Barbara, California, May 20 2001.
10. Raymond Kosala and Hendrik Blockeel. Web Mining Research: A Survey. *ACM SIGKDD Explorations*, 2(1):1–15, July 2000.
11. Shun-Tak A. Leung, Sharon E. Perl, Raymie Stata, and Janet L. Wiener. (2001). Towards Web-Scale Web Archeology. Research Report 174, Compaq Systems Research Center, Palo Alto, California. Retrieved December 10, 2003 from http://gatekeeper.dec.com/pub/DEC/SRC/research-reports/SRC-174.pdf.
12. Weiqiang Lin, Mehmet A. Orgun, and Graham. J. Williams. An Overview of Temporal Data Mining. In *Proceedings of the 1st Australian Data Mining Workshop (ADM02)*, Canberra, Australia, December 2002.
13. Sanjay Kumar Madria, Sourav S. Bhowmick, Wee Keong Ng, and Ee-Peng Lim. Research Issues in Web Data Mining. In *Proceedings of the 1st International Conference on Data Warehousing and Knowledge Discovery (DAWAK99), LNCS 1676*, pages 303–312, Florence, Italy, 1999.
14. NASA Earth Observatory. (2002). Flooding in Germany. Retrieved June 2003, from http://earthobservatory.nasa.gov/Newsroom/NewImages/images.php3?img_id=10301.

15. The Association of Bay Area Governments (ABAG). (2003). Impacts of California Earthquakes on Buildings from Shaken Awake! Retrieved December 10, 2003, from http://www.abag.ca.gov/bayarea/eqmaps/shelpop/bldg.html.
16. Barbara Pernici and Monica Scannapieco. Data Quality in Web Information Systems. In *ER 2002, 21st International Conference on Conceptual Modeling, LNCS 2503*, pages 397–413, 2002.
17. Andreas Rauber and Andreas Aschenbrenner. (2001). Part of our Culture is Born Digital - On Efforts to Preserve it for Future Generations. Internet-Zeitschrift für Kulturwissenschaften (TRANS. On-line Journal for Cultural Studies). Retrieved November 2003, from http://www.inst.at/trans/10Nr/rauber10.htm.
18. Andreas Rauber, Oliver Witvoet, Andreas Aschenbrenner, and Robert Bruckner. Putting the World Wide Web into a Data Warehouse: A DWH-based Approach to Web Analysis. In *Proceedings of the DEXA Workshop on Very Large Data Warehouses*, pages 822–826, Aix en Provence, France, September 2002. IEEE.
19. John F. Roddick and Myra Spiliopoulou. A Bibliography of Temporal, Spatial and Spatio-Temporal Data Mining Research. *ACM SIGKDD Explorations*, 1(1):34–38, June 1999.
20. Mireille Samia. Optimization of IP Multicast Networks Performance. Master's Thesis, Faculty of Natural and Applied Sciences, Department of Computer Science, Notre Dame University, Lebanon, June 2001.
21. Mireille Samia. Temporal Web Mining. In *15. GI-Workshop über Grundlagen von Datenbanken (15th GI-Workshop on the Foundations of Databases)*, pages 27–31, Tangermünde, Germany, June 2003.

SCIENTIFIC PAPERS UNIVERSITY OF LATVIA. 2004. VOLUME 672, Datbases and Information Systems
246–257 P.

Managing and Implementing the Data Mining Process Using a Truly Stepwise Approach

Perttu Laurinen[1], Lauri Tuovinen[1], Eija Haapalainen[1], Heli Junno[1], Juha Röning[1] and Dietmar Zettel[2].

[1)] Intelligent Systems Group, Department of Electrical and Information Engineering, PO BOX 4500, FIN-90014 University of Oulu, Finland. E-mail: perttu.laurinen@ee.oulu.fi.

[2]Fachhochschule Karlsruhe, Institut für Innovation und Transfer, Moltkestr. 30, 76133 Karlsruhe, Germany. E-mail: dietmar.zettel@fh-karlsruhe.de.

Abstract. Data mining consists of transformation of information with a variety of algorithms to discover the underlying dependencies. The information is passed through a chain of algorithms and usually not stored until it has reached the end of the chain, which may result in a number of difficulties. This paper presents a method for better management and implementation of the data mining process and reports a case study of the method applied to the pre-processing of spot welding data. The developed approach, called 'truly stepwise data mining', enables more systematic processing of data. It verifies the correctness of the data, allows easier application of a variety of algorithms to the data, manages the work chain, and differentiates between the data mining tasks. The method is based on storage of the data between the main phases of the data mining process. The different layers of the storage medium are defined on the basis of the type of algorithms applied to the data. The layers defined in this research consist of raw data, pre-processed data, features, and models. In conclusion, we present a systematic, easy-to-apply method for implementing and managing the work flow of the data mining process. A case study of applying the method to a resistance spot welding quality estimation project is presented to illustrate the superior performance of the method compared to the currently used approach.

Key words: hierarchical data storage, work flow management, data mining work flow implementation.

1. Introduction

Data mining consists of transformation of information with a variety of algorithms to discover the underlying dependencies. The information is passed through a chain of algorithms, and the success of the process is determined by the outcome. The typical phases of a data mining process are: raw data acquisition, pre-processing, feature extraction, and modeling. The method of managing the interactions between these phases has a major impact on the outcome of the project.

The traditional approach of implementing the data mining process is to combine the algorithms developed for the different phases and to run the data through the chain, as presented in Figure 1.

The emphasis in the approach presented in Figure 1 is on the algorithms processing the data. The information is expected to flow smoothly through the chain from the beginning to the end on a single run, and the algorithms are usually implemented within the same application. It is not unusual that the data analyst takes care of all the phases, and not much attention is always paid to the (non-standard) storage format of the data. This may result in a number of difficulties that detract from the quality of the data mining process. To name a few, the approach makes it more challenging to apply methods implemented in a variety of tools to the data, requires comprehensive knowledge from the analyst, and results in incoherent storage of the research results and data.

The approach proposed in this study has a different perspective toward implementing the data mining process. The emphasis is on a standard way of storing the data between the different phases of the process. This, in turn, increases the independence between the transformations and the data storage. Standard storage makes it possible for the algorithms to access the data through a standard interface, which allows interaction between the data and the algorithms implemented in various applications supporting the interface. The approach will be explained in more detail in the next chapter, and after that, the benefits of applying it will be illustrated with a comparison to the traditional approach and a case study.

Extensive searches of scientific databases and the World Wide Web did not bring to light similar approaches applied to the implementation of the data mining process. However, there are studies and projects on the management of the data mining process. These studies identify the main phases of the process in a manner similar to that presented in Figure 1 and give a general outline of the steps that should be kept in mind when realizing the process. One of the earliest efforts – perhaps the very earliest one – was the CRISP-DM, initiated in 1996 by three companies that proceeded to form a consortium called CRISP-DM (CRoss-Industry Standard Process for Data Mining). CRISP-DM is also the name of the process model created by the consortium, which was proposed to serve as a standard reference for all appliers of data mining [1]. The goal of the process model is to offer a representation of the phases and tasks involved that is generic enough to be applicable to any data mining effort as well as guidelines on how to apply the process model. Although it is difficult to verify a method as generic beyond all doubt, several studies testify to the usefulness of CRISP-DM as a tool for managing data mining ventures ([2], [3], [4]). The approach proposed in CRIPS-DM was extended in RAMSYS [5], which proposed a methodology for performing collaborative data mining work. Other proposals, with many similarities to CRISP-DM, for the data mining process were presented in [6] and [7]. Nevertheless, these studies did not take a stand on what would be an effective implementation of the data mining process in practice. This study proposes an effective approach for implementing the data mining process and compares it to the traditional way of implementing the process, pointing out the obvious advantages of the proposed method.

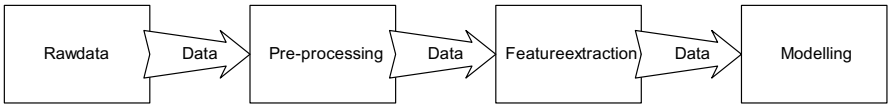


Figure 1: The traditional data mining process.

2. A truly stepwise method for managing and implementing the data mining process

This chapter presents a general framework for the proposed method and defines the way in which it can be applied to the management of the data mining process. The two basic issues that result in a number of difficulties when using the traditional approach to data mining are:

1) The transformations are organized in a way that makes them highly dependent on each other, and
2) all transformations are usually calculated at once.

To demonstrate these problems and to present an idea for a solution, the following formalism is used. The data supplied for the analysis can be assumed to be stored in a matrix $\mathbf{X}_0$. The result of the analysis, $\mathbf{X}_n$, is obtained when the *n*th transformation (function) $f_n(\mathbf{X}_0)$ is applied to the data. The transformations are applied step-by-step to the data, but they are calculated all at once, and the results are not stored until the last transformation has been applied. Using the above notation, the process can be defined as the inner functions of the supplied data, which leads to:

Definition: Stepwise data mining process (the traditional approach). The stepwise data mining process is a chain of inner transformations, $f_1 \ldots f_n$, that process the raw data, $\mathbf{X}_0$, without storing it until the desired data, the output $\mathbf{X}_n$, has been obtained:

$$\mathbf{X}_n = f_n(\ldots(f_2(f_1(\mathbf{X}_0))\ldots))$$

This points out clearly the marked dependence between the transformations and the fact that all transformations are calculated at once. However, the data is not dependent on the transformations in such a way that all transformations would have to be calculated in a single run. The result, $\mathbf{X}_n$, might equally well be generated in a truly stepwise application of the transformations, which leads to the definition of the proposed data mining process.

Definition: Truly stepwise data mining process. The results, $\mathbf{X}_1, \ldots, \mathbf{X}_n$, of each transformation, $f_1 \ldots f_n$, are stored in a storage medium before applying the next transformation in the chain to them:

$$
\begin{array}{lll}
1) & \mathbf{X}_1 & | \quad f_1 / \mathbf{X}_0 0 \\
2) & \mathbf{X}_2 & | \quad f_2 / \mathbf{X}_1 0 \\
 & 4 & \quad 4 \\
n) & \mathbf{X}_n & | \quad f_n / \mathbf{X}_{n41} 0
\end{array}
$$

This approach makes the transformations less dependent on each other: to be able to calculate the *k*th transformation (k =1,…,n), one does not need to calculate all the (k-1) transformations prior to k, but just to fetch the data, $\mathbf{X}_{k41}$, stored after the (k-1)th transformation and to apply the transformation k to that. The obvious difference between the two processes is that, in the latter, the result of the *k*th transformation is dependent only on the data, $\mathbf{X}_{k41}$, while in the former, it is dependent on $\mathbf{X}_0$ and the transformations $f_1 ... f_{k41}$. The difference between these two definitions, or approaches, might seem small at this stage, but it will be shown below how large it actually is.

In theory, the result of each transformation could be stored in the storage medium (preferably a database). In the context of data mining, however, it is more feasible to store the data only after the main phases of the transformations. The main phases are the same as those shown in Figure 1. Now that the proposed truly stepwise data mining process has been defined and the main phases have been identified, the stepwise process presented in Figure 1 can be altered to reflect the developments, as shown in Figure 2. The apparent change is the emphasis on the storage of the data. In Figure 1, the data flowed from one transformation to another, and the boxes represented the transformations. In Figure 2, the boxes represent the data stored in the different layers, and the transformations make the data flow from one layer to another. Thus, the two figures have all the same components, but the effect of emphasizing the storage of the data is apparent. The notion of the transformations carrying the data between the storage layers also seems more natural than the idea that the data is transmitted between the different transformations.

A few more comments on the diagram should be made before presenting the comparison of the two approaches. Four storage layers are defined, i.e. the layers of raw data, pre-processed data, features, and models. One more layer could be added to the structure: a layer representing the best model selected from the pool of available models. On the other hand, this is not necessary, since the presented approach could be applied to the pool of models, treating the generated models as raw data. In this case, the layers would define the required steps for choosing the best model. Another comment can be made concerning the amount and scope of data stored in the different layers. As the amount of data grows toward the bottom layers, the scope of data decreases, and vice versa. In practice, if the storage capabilities of the system are limited and unlimited amounts of data are available, the stored features may cover a broader range of data than pure data could. This is pointed out in the figure by the two arrows on the sides.

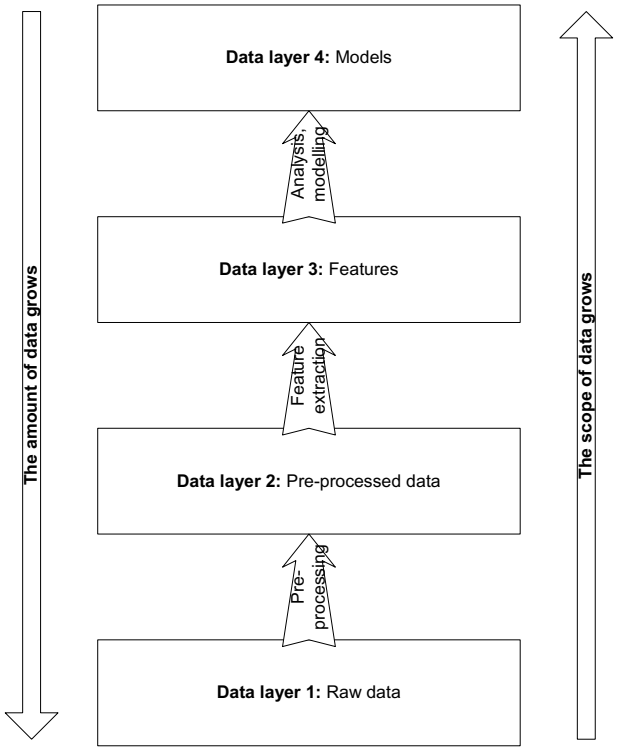


Figure 2: The four storage layers of the proposed data mining process.

3. The proposed vs. the traditional method

In this chapter, the various benefits of the truly stepwise approach over the stepwise one are illustrated.

Independence between the different phases of the data mining process. In the stepwise approach, the output of a transformation is directly dependent on each of the transformations applied prior to it. To use an old phrase, the chain is as weak as its weakest link. In other words, if one of the transformations does not work properly, none of the transformations following it can be assumed to work properly, either, since each is directly dependent on the output of the previous transformations. In the truly stepwise method, a transformation is directly dependent only on the data stored in the layer immediately prior to the transformation, not on the previous transformations. The transformations prior to a certain transformation do not necessarily have to work perfectly, it is enough that the data stored in the previous layers is correct. From the viewpoint of the transformations, it does not matter how the data was acquired, e.g. whether it was calculated using the previous transformations or even inserted manually.

The *multitude of algorithms* easily applicable to the data. In the stepwise procedure, the algorithms must be implemented in one way or another inside the

same tool, since the data flows directly from one algorithm to another. In the truly stepwise approach, the number of algorithms is not limited to those implemented in a certain tool, but is proportional to the number of tools that implement an interface for accessing the storage medium. The most frequently used interface is the database interface for accessing data stored in a database using SQL. Therefore, if a standard database is used as a storage medium, the number of algorithms is limited to the number of tools implementing a database interface – which is large.

Specialization and *teamwork* of researchers. The different phases of the data mining process require so much expertise that it is hard to find people who would be experts in all of them. It is easier to find an expert specialized in some of the phases or transformations. However, in most data mining projects, the researcher must apply or know details of many, if not all, of the steps of the data mining chain, to be able to conduct the work. This results in wasted resources, since it takes some of her / his time away from the area she / he is specialized in. Furthermore, when a team of data miners is performing a data mining project, it might be that everybody is doing a bit of everything. This results in confusion in the project management and de-synchronization of the tasks. Using the proposed method, the researchers can work on the data relevant to their specialization. When a team of data miners are working on the project, the work can be naturally divided between the workers by allocating the data stored in the different layers to suit the expertise and skills of each person.

Data storage and *on-line monitoring.* The data acquired in the different phases of the data mining process is stored in a coherent way when, for example, a standard database is used to implement the truly stepwise process. When the data can be accessed through a standard interface after the transformations, one can peek in on the data at any time during the process. This can be convenient, especially in situations where the data mining chain is delivered as a finished implementation. When using a database interface, one can even select the monitoring tools from a set of readily available software. To monitor the different phases of the stepwise process, it would be necessary to display the output of the transformations in some way, which requires extra work.

Time savings. When the data in the different layers has been calculated once in the truly stepwise process, it does not need to be re-calculated unless it needs to be changed. When working with large data sets, this may result in enormous time savings. Using the traditional method, the transformations must be recalculated when one wants to access the output of any phase of the data mining chain, which results in unnecessary waste of staff and CPU time.

Now that the numerous benefits of the proposed method have been presented, we could ask what the drawbacks of the method are. The obvious reason for the need for time is the care and effort one has to invest in defining the interface for transferring the intermediate data to the storage space. On the other hand, if this work is left undone, one may have to put twice as much time in tackling with the flaws in the data mining process. It might also seem that the calculation of the whole data mining chain using the stepwise process is faster than in the truly stepwise process. That is true, but once the transformations in the truly stepwise process are ready and finished, the process can be run in the stepwise manner. In conclusion, no obvious drawbacks are so far detectable in the truly stepwise process.

4. A case study – pre-processing spot welding data

This chapter illustrates the benefits of the proposed method in practice. The idea is here applied to a data mining project analysing the quality of spot welding joints, and a detailed comparison to the traditional approach is made concerning the amount of work required for acquiring pre-processed data.

The spot welding quality improvement project (SIOUX) is a two-year EU-sponsored CRAFT project aiming to create non-destructive quality estimation methods for a wide range of spot welding applications. Spot welding is a welding technique widely used in, for example, the electrical and automotive industries, where more than 100 million spot welding joints are produced daily in the European vehicle industry only [8]. Non-destructive quality estimates can be calculated based on the shape of the signal curves measured during the welding event [9], [10]. The method results in savings in time, material, environment, and salary costs – which are the kind of advantages that the European manufacturing industry should have in their competition against outsourcing work to cheaper countries.

The collected data consists of information regarding the welded materials, the quality of the welding spot, the settings of the welding machine, and the voltage and current signals measured during the welding event. To demonstrate the data, the left panel of Figure 3 displays a typical voltage curve acquired from a welding spot, and the right panel shows a resistance curve obtained by pre-processing the data.

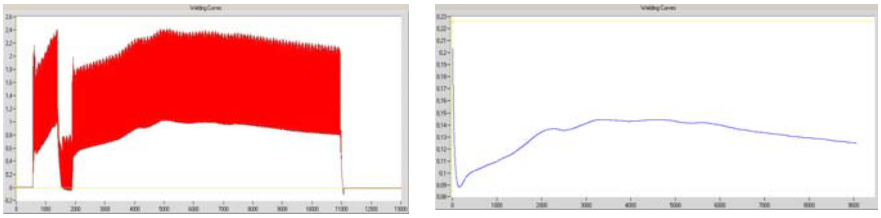


Figure 3: The left panel shows a voltage signal of a welding spot measured during a welding event. The high variations and the flat regions are still apparent in the diagram. The right panel shows the resistance curve after pre-processing.

The data transformations needed for pre-processing signal curves consist of removal of the flat regions from the signal curves (welding machine inactivity), normalization of the curves to a pre-defined interval, smoothing of the curves using a filter, and calculation of the resistance curve based on the voltage and current signals.

The transformations are implemented in software written specifically for this project, called Tomahawk. The software incorporates all the algorithms required for calculating the quality estimate of a welding spot, along with a database for storing the welding data. The software and the database are closely connected, but independent. The basic principles of the system are presented in Figure 4. The special beauty of Tomahawk lies in the way the algorithms are implemented as a connected chain. Hence, the product of applying all the algorithms is the desired output of the data mining process. The algorithms are called plug-ins, and the way the data is transmitted between each pair of plug-ins is well defined. When the

program is executed, the chain of plug-ins is executed at once. This is an implementation of the definition of the stepwise (traditional) data mining process.

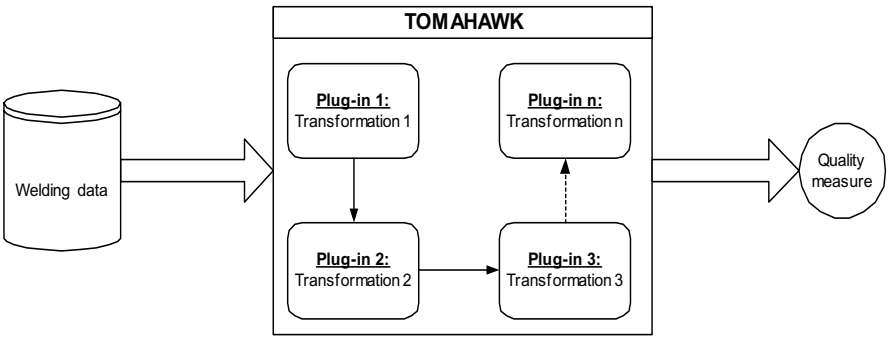


Figure 4: The operating principle of the Tomahawk software. The architecture is a realization of the stepwise data mining process.

When the project has been completed, all the plug-ins should be ready and work for all kinds of welding data as seamlessly as presented in Figure 4. However, in the production phase of the system, when the plug-ins are still under active development, three major issues that interfere with the daily work of the development team can be recognized in the chapter "The proposed vs. the traditional method".

- *Independence.* It cannot be guaranteed that all parts of the pre-processing algorithms would work as they should for all the available data. However, the researcher working on the pre-processed data is dependent on the pre-processing sequence. Because of this, she/he cannot be sure that the data is always correctly pre-processed.
- *Specialization and teamwork.* The expert working on the pre-processed data might not have the expertise to correctly pre-process the raw data in the context of Tomahawk, which would make it impossible for him/her to perform her/his work correctly.
- *The multitude of algorithms* easily applicable to the data. In the production phase, it is better if the range of algorithms tested on the data is not exclusively limited to the implementation of the algorithms in Tomahawk, since it would require a lot of effort to re-implement algorithms available elsewhere as plug-ins before testing them.

The solution was to develop Tomahawk in such a way that it supports the truly stepwise data mining process. A plug-in capable of storing and delivering pre-processed data was implemented. Figure 5 presents the effects of the developments. The left panel displays the pre-processing sequence prior to the adjustments. All the plug-ins were calculated at once, and they had to be properly configured to obtain properly pre-processed data. The right panel shows the situation after the adoption of the truly stepwise data mining process. The pre-processing can be done in its own sequence, after which a plug-in that inserts the data into the database is applied.

Now the pre-processed data is in the database and available for further use at any given time.

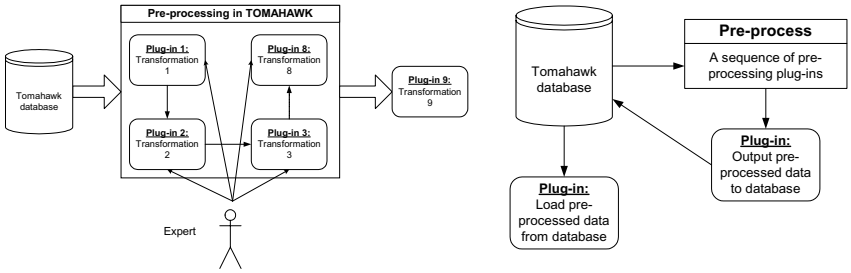


Figure 5: The left panel shows the application of the stepwise data mining process on the pre-processing of the raw data in Tomahawk. The right panel shows Tomahawk after the modifications that made it support the truly stepwise data mining process for pre-processing.

The first and second issues are simple to solve by using the new approach. The pre-processing expert of the project takes care of properly configuring the pre-processing plug-ins. If the plug-ins need to be re-configured or re-programmed for different data sets, she / he has the requisite knowledge to do it, and after the application of the re-configured plug-ins, the data can be saved in the database. If it is not possible to find a working combination of plug-ins at the current state of development, the data can still be pre-processed manually, which would not be feasible when using the stepwise process. After this, the expert in working on pre-processed data can load the data from the database and be confident that the data she / he is working on has been correctly pre-processed. The third issue is also easy to solve; after the modifications, the set of algorithms that can be tested on the data is no longer limited to those implemented in Tomahawk, but includes tools that have a database interface implemented in them, for example Matlab. This expands drastically the range of available algorithms, which in turn makes it also faster to find an algorithm suitable to a given task. As soon as a suitable algorithm has been found, it can be implemented in Tomahawk.

Finally, a comparison of the steps required for pre-processing the data in the SIOUX project using the stepwise and truly stepwise approaches is presented. The motivation of the comparison is to demonstrate how large a task it would be for the researcher working on pre-processed data to pre-process the data using the stepwise approach before she / he could start the actual work.

If one wants to acquire pre-processed data using the stepwise approach, it takes the application and configuration of 8 plug-ins to pre-process the data. The left panel of Figure 6 shows one of the configuration dialogs of the plug-ins. This particular panel has 4 numerical values that must be set correctly and the option of setting 6 check boxes. The total number of options the researcher has to set in the 8 plug-ins for acquiring correctly pre-processed data is 68. The 68 options are not the same for all the data gathered in the project, and it requires advanced pre-processing skills to configure them correctly. Therefore, it is quite a complicated task to pre-process the data, and it is especially difficult for a researcher who has not constructed the pre-

processing plug-ins. The need to configure the 68 options of the pre-processing sequence would take a lot of time and expertise away from the actual work and still give poor confidence in that the data is correctly pre-processed.

To acquire the pre-processed data using the truly stepwise approach, one only needs to fetch the data from the database. The right panel of Figure 6 shows the configuration dialog of the database plug-in, which is used to configure the data fetched for analysis from the database. Using the dialog, the researcher working on the pre-processed data can simply choose the pre-processed data items that will be used in the further analyses, and she / he does not have to bother with the actual pre-processing of the data. The researcher can be sure that all the data loaded from the database has been correctly pre-processed by the expert in pre-processing. From the viewpoint of the researcher responsible for the pre-processing, it is good to know that the sequence of pre-processing plug-ins does not have to be run every time that pre-processed data is needed, and that she / he can be sure that correctly pre-processed data will be used in the further steps of the data mining process.

In conclusion, by using the stepwise process, a researcher working with pre-processed data could never be certain that the data had been correctly pre-processed, or that all the plug-ins had been configured the way they should, which resulted in confusion and uncertainty about the quality of the data. The truly stepwise process, on the other hand, allowed a notably simple way to access the pre-processed data, resulted in time savings, and ensure that the analyzed data were correctly pre-processed.

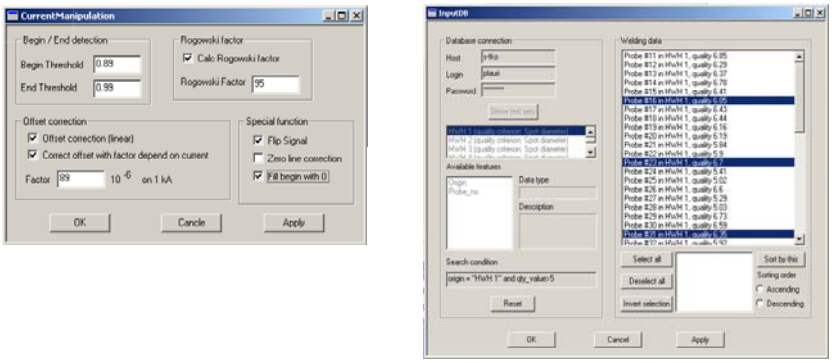


Figure 6: The left panel shows one of the 8 dialogues that need to be filled in to acquire pre-processed signal curves. The right panel shows the dialogue that is used for fetching raw and pre-processed data directly from the database.

5. Conclusions

This paper presented a new approach for managing the data mining process, called truly stepwise data mining process. In the truly stepwise process, the transformed data is stored after the main phases of the data mining process, and the transformations are applied to data fetched from the data storage medium. The benefits of the process compared to the stepwise data mining process (the traditional

approach) were analyzed. It was noticed that the proposed approach increases the independence of the algorithms applied to the data and the number of algorithms easily applicable to the data and makes it easier to manage and allocate the expertise and teamwork of the data analysts. Also, data storage and on-line monitoring of the data mining process are easier to organize using the new method, and it saves both staff and CPU time. The approach was illustrated using a case study of a spot welding data mining project. The two approaches were compared, and it was demonstrated that the proposed method markedly simplified the tasks of the specialist working on the pre-processed data.

In the future, the possibilities to apply the approach on a finer scale will be studied - here it was only applied after the main phases of the data mining process. The feature and model data of the approach will also be demonstrated, and the application of the method will be extended to other data mining projects.

6. Acknowledgements

We would like to express our gratitude to our colleagues at Fachochschule Karlsruhe, Institut für Innovation und Transfer, in Harms + Wende GmbH & Co.KG [11], in Technax Industrie [12] and in Stanzbiegetechnik GesmbH [13] for providing the data set, the expertise needed in the case study and for numerous other things that made it possible to accomplish this work. We also wish to thank the graduate school GETA [14], supported by Academy of Finland, for sponsoring this research. Furthermore, this study has been financially supported by the Commission of the European Communities, specific RTD programme “Competitive and Sustainable Growth”, G1ST-CT-2002-50245, “SIOUX” (Intelligent System for Dynamic Online Quality Control of Spot Welding Processes for Cross(X)-Sectoral Applications”). It does not necessarily reflect the views of this programme and in no way anticipates the Commission’s future policy in this area.

References

[1] P. Chapman, J. Clinton, T. Khabaza, T. Reinartz and R. Wirth, "CRISP-DM 1.0 Step-by-step data mining guide," August, 2000.

[2] Hotz, E., Grimmer, U. Heuser, W. & Nakhaeizadeh, G. 2001. REVI-MINER, a KDD-Environment for Deviation Detection and Analysis of Warranty and Goodwill Cost Statements in Automotive Industry. In Proc. Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD 2001), 432–437.

[3] Liu, J.B. & Han, J. 2002. A Practical Knowledge Discovery Process for Distributed Data Mining. In Proc. ISCA 11th International Conference on Intelligent Systems: Emerging Technologies, 11–16.

[4] Silva, E.M., do Prado, H.A. & Ferneda, E. 2002. Text mining: crossing the chasm between the academy and the industry. In Proc. Third International Conference on Data Mining, 351–361.

[5] S. Moyle and A. Jorge, "RAMSYS - A methodology for supporting rapid remote collaborative data mining projects," in ECML/PKDD'01 workshop on Integrating

Aspects of Data Mining, Decision Support and Meta-Learning: Internal SolEuNet Session, 2001, pp. 20-31.

[6] D. Pyle, Data Preparation for Data Mining, Morgan Kaufmann Publishers, 1999.

[7] R.J. Brachman and T. Anand, "The Process of Knowledge Discovery in Databases: A Human-Centered Approach," in Advances in Knowledge Discovery and Data Mining, U.M. Fayyad, G. Piatetsky-Shapiro, P. Smyth and R. Uthurusamy Eds. MIT Press, 1996, pp. 37-58.

[8] TWI World Centre for Materials Joining Technology, information available at their homepage: http://www.twi.co.uk/j32k/protected/band_3/kssaw001.html, referenced 31.12.2003.

[9] Laurinen, P.; Junno, H.; Tuovinen, L.; Röning, J.; Studying the Quality of Resistance Spot Welding Joints Using Bayesian Networks, Artificial Intelligence and Applications (AIA 2004), February 16-18, 2004, Innsbruck, Austria.

[10] Junno, H.; Laurinen, P.; Tuovinen, L.; Röning, J.; Studying the Quality of Resistance Spot Welding Joints Using Self-Organising Maps, Fourth International ICSC Symposium on Engineering of Intelligent Systems (EIS 2004), February 29 -March 2, 2004, Madeira, Portugal.

[11] Harms+Wende GmbH & Co.KG, the world wide web page: http://www.harms-wende.de/, referenced 13.2.2004.

[12] Technax Industrie, the world wide web page: http://www.technaxindustrie.com/, referenced 13.2.2004.

[13] Stanzbiegetechnik GesmbH, the world wide web page: http://www.stanzbiegetechnik.at/Startseite/index.php, referenced 13.2.2004.

[14] Graduate school GETA, the world wide web page: http://wooster.hut.fi/geta/, referenced 13.2.2004.

SCIENTIFIC PAPERS UNIVERSITY OF LATVIA. 2004. VOLUME 672, Datbases and Information Systems
258–269 P.

Association Rule Mining Meets Functional Dependencies: The AP-FD Algorithm

Jürgen M. Janas

Fakultät für Wirtschafts- und Organisationswissenschaften
Universität der Bundeswehr München
Werner-Heisenberg-Weg 39, D-85577 Neubiberg, Germany
juergen.janas@unibw-muenchen.de

Abstract. Association rule mining has mainly been studied in terms of transactional data such as the market basket example; comparatively little attention has been paid to its generalization to arbitrary relational data. Although this is justified to some extent because the methods applied to transaction data may easily be adapted to relational data, there are certain peculiarities of relational data that cannot be taken advantage of by these adapted methods. In this paper, we examine the role which functional dependencies – a concept which has been studied rigorously in the area of relational database design – may play in mining relational data for association rules. We will show how the knowledge of functional dependencies may be used to improve the performance of the A Priori algorithm which is the most popular algorithm for finding sets of frequently co-occurring attribute values.

Keywords. Data mining, association rules, A Priori algorithm, relational databases, functional dependencies

1. Introduction

Data mining is the analysis of typically very large sets of data in order to discover yet unknown relationships among the data and aggregate them in ways which are novel, useful, and understandable to the users of the data. Data mining is a still young and due to its different ancestors, namely statistics, databases and artificial intelligence, heterogeneous discipline. Among its sub-disciplines, association rule mining is probably the one which is most independent from these ancestors.

Association rule mining (originally introduced in [1]) is usually explained in the context of the market-basket problem which is the task of identifying sets of items which frequently occur together in supermarket transaction data. Technically spoken, the rules to be mined in this scenario are single-dimensional association rules, i.e. both sides of these rules are sets of values which come from the same domain, in this case the items that may be purchased in the supermarket. Single-dimensional association rules may be generalized quite naturally to multidimensional association rules which are better tailored to data according to the relational model of data.

Association rule mining is usually done in two phases. During the first phase, one of the numerous variants of an algorithm which is known as “A Priori” is used to

determine sets of attribute values which frequently occur together by making a number of sequential scans of the data to be mined. During the second phase, the sets of attribute values found in the first phase are split into the left hand side and the right hand side of one or more association rules according to some simple statistical criteria; it is assumed that these rules might be interesting to the user and therefore they are presented to him for inspection.

In this paper, we will propose an improvement to the first phase of association rule mining that uses knowledge about the functional dependencies that are contained in the data to be mined. Functional dependencies are patterns in data according to the relational model of data that may be observed due to corresponding regularities among the real world objects which are to be modeled by the data; they are commonly employed in the process of database design.

We will show that the knowledge of functional dependencies in the data to be mined allows us to logically infer that certain sets of attribute values occur frequently together without counting their occurrences. This observation is used in a new variant of the A Priori algorithm, namely the AP-FD algorithm (which owes its name to its two main ingredients, i.e. the classical A Priori algorithm and functional dependencies), which is superior to the original algorithm with respect to both runtime and space requirements.

The remainder of this paper is organized as follows. Section 2 serves to introduce the concepts and the notation used within this paper; this comprises concepts both from relational database theory and from data mining; in particular, an adaptation of the A Priori algorithm to multidimensional association rules is presented. The similarities and the differences of functional dependencies on the one hand and association rules on the other as well as the interactions between these two concepts will be investigated in section 3; moreover, we will use this section for a brief discussion of the question to what extent it is reasonable to expect that functional dependencies will be present in the data to be mined for multidimensional association rules. The fourth section introduces the AP-FD algorithm and contains a discussion of its performance. The final section is used to summarize the results of the paper.

2. Basic concepts and notation

2.1. Relations and functional dependencies

We assume that the reader is familiar with the basic concepts of the relational model of data (e.g. on the basis of [2]) and restrict ourselves to informally introducing the concepts we are using in the rest of this paper. When talking about a **relation** we distinguish between the relation schema (denoted by R) which describes the structure of the relation and the relation instance (denoted by r) which contains the stored data and conforms to the structure given by the relation schema.

A **relation schema** R comprises the set of **attributes** (denoted by atts(R)) on which R is defined and a set of **integrity constraints** which have to be satisfied by a relation instance in order to be conforming to R. We will use letters X and Y for sets of attributes and letters A and B for individual attributes.

A **relation instance** may be thought of as a set of **rows** each of which consists of a set of values such that for each of the attributes in atts(R) there is exactly one corresponding value; individual such values will be referred to as **attribute values** and be denoted by A:a where a stands for the value corresponding to the attribute A. A **set of co-occurring attribute values** (abbreviated **scav**) consists of one attribute value per attribute from some attribute set X ≥ atts(R) and will be designated by X:x. We shall not make a distinction between an attribute value and the scav which contains only this attribute value.

The notation $\omega_{X:x}(r)$ stands for the relation instance which is obtained by selecting those rows from r that contain the scav X:x. Finally, card(r) is used to designate the number of rows contained in the relation instance r.

As far as the integrity constraints of a relation schema are concerned, we restrict our considerations to functional dependencies within this paper. A **functional dependency** is an expression of the form X ⇓ A. We say **X ⇓ A is satisfied by a relation instance** r if and only if every two rows from r which agree with respect to their values for X, also agree with respect to their values for A; moreover, we say that **X ⇓ A holds in a relation schema** R if and only if X ⇓ A is satisfied by every relation instance conforming to R.

A functional dependency X ⇓ A is called a **trivial functional dependency** if and only if A ⊂ X. Obviously, all trivial functional dependencies hold in the respective relation schema.

With regard to the functional dependencies which hold in a relation schema R we distinguish between the set B of functional dependencies given as part of the relation schema and the set B^+ of all functional dependencies that may be inferred from B. B^+ always contains all functional dependencies from B and all trivial functional dependencies, but it may contain additional functional dependencies which may be derived from B according to transitivity and other rules (for details see [2]).

A **key** of a relation schema R is a minimal set X ≥ atts(R) such that the functional dependency X⇓ A holds in R for each A ⊂ atts(R).

The **closure of a set of attributes** X with respect to B is the set of all attributes A such that X ⇓ A is contained in B^+; it will be denoted by X_B^+. The closure o a set of attributes can be comuted in linear time; a corresponding algorithm may may be found in [3].

2.2. Association rules

If R is a relation schema, X ≥ atts(R), and A ⊂ atts(R), then an expression of the form

$$(X{:}x \ \blacktriangledown \ A{:}a, s, c)$$

where s and c are real numbers from the interval between 0 and 1 is called a **multidimensional association rule**; X:x is called the left hand side and A:a the right hand side of the rule. If r is a relation instance conforming to R, we say that the multidimensional association rule (X:x ♥ A:a, s, c) is satisfied by r if

$$\mathrm{card}(\omega_{A:a}(\omega_{X:x}(r))) / \mathrm{card}(r) \ \acute{O} s$$

and

$$\mathrm{card}(\omega_{A:a}(\omega_{X:x}(r))) / \mathrm{card}(\omega_{X:x}(r)) \geq c$$

Both s and c are measures for the interestingness of a multidimensional association rule; s is called the **support threshold** and prescribes a lower bound for the fraction of the rows of r that have to contain both the left hand side and the right hand side of the rule for the rule to be satisfied by r.

The **confidence level** c prescribes at least what fraction of the rows of r that contain the left hand side of the rule have to contain the right hand side as well for the rule to be satisfied by r.

Multidimensional association rules are particularly appropriate for mining data according to the relational model of data. For reasons of brevity, multidimensional association rules will be referred to as association rules throughout the rest of this paper.

2.3. The classical A Priori algorithm

Mining of association rules is usually done in two phases: During the first phase, the **frequent scavs** are determined, i.e., those scavs which occur in at least that fraction of the rows that is specified by the support threshold. During the second phase, the frequent scavs are split into a left hand side X:x and a right hand side A:a according to additional criteria such as a lower bound for the confidence level of the resulting rule.

The most popular algorithm for the first phase is the so-called A Priori algorithm which was originally introduced in [4]. The A Priori algorithm is based on the observation that if a scav appears in a fraction s of the rows of a relation instance, then any subset of this scav appears in at least fraction s of the rows, or – to put it the other way round – a scav may appear in a fraction s of the rows only if each of its subsets does so. This observation is sometimes referred to as the **A Priori trick**.

In the relevant literature, the A Priori algorithm is usually explained in the context of the market-basket problem which implies a restriction to single-dimensional association rules. However, as observed in [5], the algorithm may easily be adapted to multidimensional association rules. In Fig. 1, the classical A Priori algorithm is restated on a rather abstract level and in such a way that it applies to multidimensional association rules; thus we make it directly comparable to the algorithm which will be proposed in section 4 of this paper and which is applicable to multidimensional association rules only. The algorithm takes as input a relation instance r and a support threshold s and outputs the set L of all frequent scavs.

The algorithm proceeds levelwise and requires one pass per level through the given relation instance. On level k, the set $\mathbf{L_k}$ of all frequent scavs of size k is determined; this is done in two steps: During the first step, a set $\mathbf{C_k}$ of candidate scavs is constructed from $\mathbf{L_{k-1}}$ by applying the A Priori trick. In the second step, the occurrences of the candidate scavs in the relation instance are counted and thus the actually frequent ones are identified. For further details and a discussion of how to compute the candidate sets $\mathbf{C_k}$ efficiently, the reader is referred to [5].

```
begin
    C1 := {{A:a}| r contains an occurrence of A:a};
    L1 := {{A:a}| {A:a} ⊂ C1 ⟨ card(ωA:a(r))Ós};
    L := ∴;
    k := 1;
    while Lk Í ∴ do
        begin
        L := L ≅ Lk;
        k := k + 1;
        Ck := {C| C = {A1:a1,...,Ak:ak} ⟨ AiÍAj for 1Òi<jÒk
                                ⟨ C\{Ai:ai} ⊂ Lk-1 for 1ÒiÒk};
        Lk := {X:x| X:x ⊂ Ck ⟨ card(ωX:x(r))Ós}
        end
end.
```

Figure 1. The classical A Priori algorithm

3. A common view on association rules and functional dependencies

3.1. Technical interactions

At first glance, association rules and functional dependencies seem to show a good deal of similarities: As a matter of fact, both of them are meant to characterize situations in which the values with respect to one or more attributes of a relation determine the values with regard to some other attribute of that relation. However, there are three aspects in which association rules and functional dependencies differ from each other considerably.

First of all, functional dependencies have a coarser granularity than association rules. This means, while a functional dependency X ⇓ A is a statement which refers to the entirety of the values with regard to X and to the entirety of the values with regard to A in a relation instance, an association rule (X:x ♥ A:a, s, c) is a statement which relates to the combination of only one particular combination of values with regard to X and one particular value with regard to A.

Secondly, functional dependencies are more stringent than association rules; that is, while a functional dependency enforces that all of the rows which contain the same values with regard to X have to agree with respect to their value for A, an association rule requires only that a certain percentage (expressed by means of the confidence level c) of the rows which contain the values with regard to X have the same value with regard to A.

Finally, functional dependencies relate to the relation schema, whereas association rules relate to a specific relation instance. This implies that a functional dependency will be satisfied by every relation instance which conforms to the respective relation schema; thus the validity of a functional dependency cannot be affected by changes to the relation instance. An association rule, by way of contrast,

is a statement about one particular relation instance only and therefore such a statement may become invalid due to a change to the respective relation instance.

It is an immediate consequence of the latter of these differences between association rules and functional dependencies that the existence of one or more association rules will by no means have any effect on the set of functional dependencies which hold in the respective relation schema.

On the other hand, a functional dependency $X \Downarrow A$ which holds in a relation schema implies the validity of all association rules (X:x ♥ A:a, s, c) for those values with regard to X and to A which occur in combination in the respective relation instance; this implication is independent of the respective relation instance and the confidence level c of each of the implied association rules is equal to 1. Of course, this kind of relationship is owed to the different granularity of functional dependencies and association rules and it is of a rather trivial nature.

The following lemma will show, however, that there is another, more useful kind of interaction between functional dependencies and association rules. As a matter of fact, this interaction is not expressed in terms of association rules, but rather in terms of frequent scavs from which the association rules that are satisfied by a particular relation instance may be derived.

Lemma. Let R be a relation schema, r a relation instance conforming to r, and $Y \geq X \geq atts(R)$; if X:x is a frequent scav then $X:x \cong Y_B^+:y'$ is also a frequent scav with $Y_B^+:y'$ being the (uniquely determined) attribute values which are contained in all rows of r which contain X: x.

Proof: obvious

3.2. Functional dependencies in the data to be mined

Before we will turn to the implications the above lemma has for finding frequent scavs, we have to address a much more general question: Functional dependencies are a concept which is commonly used in the design of relational databases; therefore, it is legitimate to ask whether we may reasonably expect that functional dependencies will be present at all in sets of data to be mined for association rules. It may at first glance seem as if the answer to this question is completely dependant on the respective data set, however, a closer look reveals that certain answers of a more general nature can be given.

Functional dependencies are used in relational database design to define certain normal forms of relation schemes that are considered desirable because they avoid redundancy in the conforming relation instances. The best normal form which may be achieved – as long as only functional dependencies are regarded – is the so-called Boyce/Codd normal form (BCNF); according to the definition of BCNF, if $X \Downarrow A$ is a non-trivial functional dependency that holds in a relation schema which is in BCNF then X has to contain a key of that relation schema.

It is easy to see that any such functional dependency $X \Downarrow A$ will be of hardly any help for mining frequent scavs because any scav which comprises all of the attributes from X will occur at most once in a conforming relation instance and therefore will hardly ever be considered frequent. As a consequence, functional dependencies hardly ever will be useful for mining associations between multiple

attribute values which are contained in the same relation, particularly if the relation schema is in BCNF.

The situation is different, however, as soon as associations between attribute values from multiple relations are considered. In this case, the relations which contain the attribute values have to be joined as a prerequisite to counting the co-occurrences of the attribute values. For reasons of brevity we restrict ourselves to the case where only two relation schemes R_1 and R_2 have to be joined.

The attribute(s) on which the join of R_1 and R_2 is performed must be a key in at least one of the relation schemes R_1 and R_2 because otherwise the join would be "lossy", i.e., the relation resulting from the join would not comprise the same information as R_1 and R_2 (cf. [2] for a thorough discussion of that matter). On the other hand, if the join attribute(s) constitute a key in R_1 it is very unlikely that they are a key of R_2 as well because in that case there would have been no need to separate the information contained in the two relations during database design. As a consequence, the left hand side of any functional dependency which holds in R_1 will not contain a key of the joined relation even if it contains a key of R_1 and therefore, a scav with attributes from $X \geq atts(R_1)$ may be contained in the joined relation an arbitrary number of times and, in particular, may be frequent. So all the functional dependencies that hold in R_1 may be useful for mining multidimensional association rules even if their left hand sides contain a key of R_1.

It is obvious that the number of functional dependencies in the data to be mined increases the number of relations which result from the normalization process and therefore increases the number of joins that are required to build the relation in which the scavs may be counted.

Apart from that, we would like to point out that the data to be mined may also contain functional dependencies for other reasons; we only mention dimension tables in data warehouses which are deliberately denormalized (cf. [6]).

4. The AP-FD algorithm

In [7], we proposed an enhanced A Priori algorithm which uses the knowledge about functional dependencies in the data to be mined essentially in the following way: If there is a frequent scav X:x in $\mathbf{L_{k-1}}$ and if a functional dependency $Y \Downarrow A$ with $Y \geq X$ and $A \supset X$ holds in R then the set $X:x \cong A:a$ is included in the set $\mathbf{L_k}$ without prior counting its occurrences in r. The main advantage of this algorithm over the classical A Priori algorithm is the fact that it avoids unnecessary inclusions of candidates into the sets $\mathbf{C_k}$. Since the size of the candidate sets $\mathbf{C_k}$ (particularly $\mathbf{C_2}$) is the bottleneck of the A Priori algorithm (cf. [8]), the enhanced A Priori algorithm is able to cope with larger data sets than the classical A Priori algorithm.

As far as runtime is concerned the classical A Priori algorithm and the algorithm proposed in [7] are of equal standard. This is because both algorithms compute the sets $\mathbf{L_k}$ of all frequent scavs of size k strictly one after another and thus require a complete scan of the data for each k. Therefore, the number of scans of the data is equal to the size of the largest frequent scav and therefore the same for both the classical and the enhanced A Priori algorithm.

As a matter of fact, the lemma stated at the end of section 3.1 may be used to speed up the computation of all frequent scavs in the presence of functional dependencies. This is because whenever we know that X:x is a frequent scav we may conclude without further counting that X_B^+:x' (x' stands for the uniquely determined values a tuple contains with regart to the attributes from X_B^+ if it contains X:x) is also a frequent scav irrespective of how many attributes are contained in $X_B^+ \backslash X$. Thus it is possible that the largest scavs will be found after a fewer number of scans of the data.

Like the classical and the enhanced A Priori algorithm, the AP-FD algorithm alternately computes a set $\mathbf{C_k}$ of candidates and a set $\mathbf{L_k}$ of frequent scavs. However, unlike the other two algorithms the scavs contained in $\mathbf{L_k}$ may be of different size; more precisely, a scav contained in $\mathbf{L_k}$ is at least of size k and consists of two parts, namely a "counting-part" and an "fd-part". The **counting-part** of a scav in $\mathbf{L_k}$ is a scav of fixed size k and the **fd-part** of a scav in $\mathbf{L_k}$ is the set of attribute values which are functionally dependent on the attribute values in the counting-part of the same scav; the fd-part may contain any number of attribute values, particularly zero.

The separation of counting-part and fd-part of a scav is made in order to be able to apply the classical A Priori algorithm to the counting-parts which are of equal size within a single set $\mathbf{L_k}$. The fd-parts are used to avoid the generating of candidates for which one may infer that they are frequent because of the given functional dependencies. For that reason a scav C will become a candidate only if none of its attribute values is contained in the fd-parts of a subset of C.

The AP-FD algorithm is given in Fig. 2; it requires – compared to the classical A Priori algorithm –a set of functional dependencies as an additional input. Note that the fd-parts of the scavs in $\mathbf{L_k}$ are not given explicitly, rather, only their attributes are mentioned because the actual values are uniquely determined by the counting-part of the respective scav anyway. Where we need to refer to such unknown values (namely in the construction of the output set **L**) in the AP-FD algorithm we do so by using the symbol "$*$".

```
begin
    C1 := {{A:a}| r contains an occurrence of A:a};
    L1 := {({A:a}, AB+\{A})| {A:a}⊂C1 ⟨ card(ωA:a(r))Ós};
    L := ∴;
    k := 1;
    while Lk Í ∴ do
        begin
        L := L ≅ {(X:x ≅ Y:*)| (X:x, Y) ⊂ Lk};
        k := k + 1;
        Ck := {C| C = {A1:a1,...,Ak:ak} ⟨ AiÍAj for 1Òi<jÒk
                        ⟨ &i ⊂ {1,...,k} )(X:x, Y) ⊂ Lk-1
                              (X:x = C\{Ai:ai} ⟨ Ai ⊃ Y)};
        Lk := {(X:x, XB+\X)| X:x⊂Ck ⟨ card(ωX:x(r))Ós}
        end
end.
```

Figure 2. The AP-FD algorithm

It is easy to see from the way $\mathbf{C_k}$ is built that there will never be any functional dependency among the attribute values which are contained in a candidate scav and as a consequence there will also never be any functional dependency among the attribute values which are contained in the counting-part of a frequent scav. Thus, the AP-FD algorithm generates as few candidates as possible when building a candidate set $\mathbf{C_k}$; in this respect the AP-FD algorithm is equal to the enhanced A Priori algorithm described in [7].

However, the AP-FD algorithm is superior to both the classical A Priori algorithm and the enhanced A Priori algorithm with regard to the number of scans of the data that are required to find a frequent scav of size k. Both the classical A Priori algorithm and the enhanced A Priori algorithm require exactly k scans under any circumstances, whereas the number of scans that are required by the AP-FD algorithm is equal to the number of attributes contained in the counting-part of the scav. Thus the overall number of scans required by the AP-FD algorithm will be reduced whenever the largest ones of the frequent scavs have functional dependencies among their attribute values.

Example. Let R be a relation schema with atts(R) = {A, B, C, D, E} and let B = {A ⇓ B, BC ⇓ D, E ⇓ C} be the set of functional dependencies which hold in R. It is easy to verify that the relation instance r given in Fig. 3 is conforming to R.

A	B	C	D	E
a_1	b_1	c_1	d_1	e_1
a_1	b_1	c_2	d_2	e_2
a_2	b_1	c_1	d_1	e_3
a_3	b_2	c_2	d_1	e_2
a_3	b_2	c_2	d_1	e_2

Figure 3. Example relation instance

We compare the performance of the classical A Priori algorithm and the AP-FD algorithm by applying both of them to r with s = 0.4 as support threshold. It is easy to see from looking at the two bottom rows of r that the classical A Priori algorithm will require five scans of r until it discovers the largest frequent scav, namely {A:a3, B:b2, C:c2, D:d1, E:e2}.

The AP-FD algorithm, on the other hand, generates the following sets one after the other:

$\mathbf{C_1}$ = { {A:a_1}, {A:a_2}, {A:a_3}, {B:b_1}, {B:b_2}, {C:c_1}, {C:c_2}, {D:d_1}, {D:d_2}, {E:e_1}, {E:e_2}, {E:e_3} }

$\mathbf{L_1}$ = { ({A:a_1}, {B}), ({A:a_3}, {B}), ({B:b_1}, ∴), ({B:b_2}, ∴), ({C:c_1}, ∴), ({C:c_2}, ∴), ({D:d_1}, ∴), ({E:e_2}, {C}) }

$\mathbf{C_2}$ = { {A:a_1, C:c_1}, {A:a_1, C:c_2}, {A:a_1, D:d_1}, {A:a_1, E:e_2}, {A:a_3, C:c_1}, {A:a_3, C:c_2}, {A:a_3, D:d_1}, {A:a_3, E:e_2}, {B:b_1, C:c_1}, {B:b_1, C:c_2}, {B:b_1, D:d_1}, {B:b_1, E:e_2}, {B:b_2, C:c_1}, {B:b_2, C:c_2}, {B:b_2, D:d_1}, {B:b_2, E:e_2}, {C:c_1, D:d_1}, {C:c_2, D:d_1}, {D:d_1, E:e_2} }

$\mathbf{L_2}$ = { ({A:a_3, C:c_2}, {B, D}), ({A:a_3, D:d_1}, {B}), ({A:a_3, E:e_2}, {B, C, D}), ({B:b_1, C:c_1}, {D}),

$(\{B{:}b_1, D{:}d_1\}, \therefore)$, $(\{B{:}b_2, C{:}c_2\}, \{D\})$,
$(\{B{:}b_2, D{:}d_1\}, \therefore)$, $(\{B{:}b_2, E{:}e_2\}, \{C, D\})$,
$(\{C{:}c_1, D{:}d_1\}, \therefore)$, $(\{C{:}c_2, D{:}d_1\}, \therefore)$,
$(\{D{:}d_1, E{:}e_2\}, \therefore)$ }

Note that during the third traversal of the while-loop the largest frequent scav in r, namely {A:a3, B:b2, C:c2, D:d1, E:e2} is already inserted into the set L; moreover, it turns out that $\mathbf{C_3}$ is the empty set and therefore the AP-FD algorithm terminates. Thus in this case, our algorithm requires only two full table scans whereas the classical A Priori algorithm requires five.

A detailed comparison of the size of the sets generated by the two algorithms is shown in Fig. 4.

	C_1	L_1	C_2	L_2	C_3	L_3	C_4	L_4	C_5	L_5
A Priori algorithm	12	8	25	14	11	11	5	5	1	1
AP-FD algorithm	12	8	19	11	–	–	–	–	–	–

Figure 4. Size of the sets generated by the classical A Priori algorithm and the AP-FD algorithm

As one can see from Fig. 4, the result of the classical A Priori algorithm and the AP-FD algorithm are not quite the same: The classical A Priori algorithm collects definitely all frequent scavs in **L**, whereas the AP-FD algorithm inserts a frequent scav into **L** only, if there is no superset of that scav which is already known to be frequent and therefore can be inserted into **L** at the same time. It is obvious that this discrepancy could be overcome easily if required.

Finally, we would like to point out that the potential of the AP-FD algorithm to reduce the number of candidate sets grows with the number of frequent scavs X:x which exist for a given X. It is easy to see that in this respect, the above example will be outperformed by real life relations, particularly, if they contain a large number of tuples.

5. Conclusion

In this paper, we have investigated the relationship between functional dependencies and multidimensional association rules. It turns out that the knowledge of functional dependencies in the data to be mined may be used in the A Priori algorithm which is commonly used to determine all frequent sets of co-occurring attribute values. For that purpose, we have proposed a new variant of the A Priori algorithm called the AP-FD algorithm. This algorithm is superior to its classical counterpart with respect to both runtime and the number of candidate sets that are generated. In particular, this latter point is of great importance as the number of candidate sets generated is the bottleneck for the applicability of the A Priori algorithm to very large data sets.

There have been numerous proposals in the literature that focus on improving the efficiency of the classical A Priori algorithm. It seems worth mentioning that the AP-FD algorithm described in this paper has not to be understood as an alternative to such proposals, but rather may be combined with many of these proposals thus

resulting in a further gain in performance. We only mention the most important ones of these variations of the classical A Priori algorithm.

Hash-based techniques (such as [9]) use one or more hash tables the entries of which are counters. While computing $\mathbf{L}_k$ from $\mathbf{C}_k$, these algorithms map all scavs of size k+1 to the buckets of the hash table and increment the corresponding counter; a scav of size k+1 then has to be inserted into $\mathbf{C}_{k+1}$ only if the corresponding counter exceeds the support threshold. It is obvious that this procedure is equally applicable if the AP-FD algorithm is employed instead of the classical A Priori algorithm.

Transaction reduction methods (as described in [4]) make use of the observation that a tuple which does not contain a frequent scav of size k cannot contain a frequent scav of size k+1. Therefore all such tuples are ruled out from the scan of the data during which $\mathbf{L}_{k+1}$ is determined by counting. Clearly this may be done as well if the AP-FD algorithm is used instead of the classical A Priori algorithm.

The **partitioning technique** (cf. [10]) applies the classical A Priori algorithm to partitions of the data to be mined that are sufficiently small in order to fit into main memory. The union of all scavs which are frequent in at least one partition is then used in a second scan of the data to find out which ones are frequent with respect to the entire data set. Clearly the AP-FD algorithm may be used to speed up the first phase of this procedure.

Sampling techniques (see [11] for an example) mine a subset of the given data for frequent scavs and then verify those scavs with respect to the entire data set by means of one or more full scans of the data. Obviously, the classical A Priori algorithm may be replaced by the AP-FD algorithm when mining the samples.

As a matter of fact, there are other methods for mining frequent scavs that do not directly build upon A Priori; we only mention DIC (for "dynamic itemset counting", described in [12]) and CARMA (for "continuous association rule mining algorithm", cf. [13]). Although one cannot incorporate the AP-FD algorithm into these methods, they may well benefit from the knowledge of functional dependencies. This is because both DIC and CARMA – as well as most other research on the subject – "are variations of the 'bottom-up theme' proposed by the A Priori algorithm" as it was called in [14]. "Bottom-up theme" means that the frequent scavs are obtained by starting with singleton scavs and then incrementally generating larger and larger scavs. Without looking at the details of such a procedure it is obvious that the knowledge of functional dependencies will always offer opportunities to overleap certain steps. How this can be done efficiently in detail, however, requires further study.

References

[1] Agrawal R., Imielinski T., Swami T. Mining Association Rules between Sets of Items in Large Databases. Proc. ACM SIGMOD Int. Conf. on Management of Data, 1993. p. 207 – 216

[2] Garcia-Molina H., Ullman J.D., Widom J. Database Systems: The Complete Book. Upper Saddle River, NJ. Prentice Hall, 2002

[3] Beeri C., Bernstein P.A: Computational Problems Related to the Design of Normal Form Relation Schemas. ACM Trans. on Database Systems 1979; 4(1): p. 30 – 59

[4] Agrawal R., Srikant R. Fast Algorithms for Mining Association Rules. Proc. Int. Conf. Very Large Data Bases, 1994. p. 487 – 499
[5] Han J., Kamber M. Data Mining: Concepts and Techniques. San Francisco. Morgan Kaufmann Publishers, 2001
[6] Kimball R. The Data Warehouse Toolkit - Practical Techniques for Building Dimensional Data Warehouses. New York. Wiley, 1996
[7] Janas J.M. An Enhanced A Priori Algorithm for Mining Multidimensional Association Rules. Proc. 25th Int. Conf. Information Technology Interfaces, 2003 p. 193 – 198
[8] Ullman J.D. Data Mining Lecture Notes. http://www-db.stanford.edu/~ullman/mining/mining.html [03/06/2003]
[9] Park J.S., Chen M.S., Yu P.S. An Effective Hash-Based Algorithm for Mining Association Rules. Proc. ACM SIGMOD Int. Conf. on Management of Data, 1995. p. 175 – 186
[10] Savasere A., Omiecinski E., Navathe S. An Efficient Algorithm for Mining Association Rules in Large Databases. Proc. Int. Conf. Very Large Data Bases, 1995. p. 432 - 443
[11] Toivonen H. Sampling Large Databases for Association Rules. Proc. Int. Conf. Very Large Data Bases, 1996. p. 134 – 145
[12] Brin S., Motwani R., Ullman J.D., Tsur S. Dynamic Itemset Counting and Implication Rules for Market Basket Analysis. Proc. ACM SIGMOD Int. Conf. on Management of Data, 1997. p. 255 – 264
[13] Hidber C. Online Association Rule Mining. Proc. ACM SIGMOD Int. Conf. on Management of Data, 1999. p.145 – 156
[14] Aggarwal C.C., Yu P.S. Mining Large Itemsets for Association Rules. Bulletin of the IEEE Computer Society TC on Data Engineering, March 1998. p. 23 – 31

SCIENTIFIC PAPERS UNIVERSITY OF LATVIA. 2004. VOLUME 672, Datbases and Information Systems
270–281 P.

Data Mining Query Scheduling for Apriori Common Counting*

Marek Wojciechowski, Maciej Zakrzewicz

Poznan University of Technology
Institute of Computing Science
ul. Piotrowo 3a, 60-965 Poznan, Poland
{marek, mzakrz}@cs.put.poznan.pl

Abstract. In this paper we consider concurrent execution of multiple data mining queries. If such data mining queries operate on similar parts of the database, then their overall I/O cost can be reduced by integrating their data retrieval operations. The integration requires that many data mining queries are present in memory at the same time. If the memory size is not sufficient to hold all the data mining queries, then the queries must be scheduled into multiple phases of loading and processing. We discuss the problem of data mining query scheduling and propose a heuristic algorithm to efficiently schedule the data mining queries into phases.

Keywords. Data mining, data mining queries

1. Introduction

Data mining is a database research field that aims at the discovery of trends, patterns and regularities in very large databases. We are currently witnessing the evolution of data mining environments towards their full integration with DBMS functionality. In this context, data mining is considered to be an advanced form of database querying, where users formulate declarative *data mining queries*, which are then optimized and executed by one of data mining algorithms built into the DBMS. One of the most significant issues in data mining query processing is long execution time, ranging from minutes to hours.

One of the most popular pattern types discovered by data mining queries are *frequent itemsets*. Frequent itemsets describe co-occurrences of individual items in sets of items stored in the database. An example of a frequent itemset can be a collection of products that customers typically purchase together during their visits to a supermarket. Such frequent itemset can be discovered in the database of customer shopping baskets. Frequent itemsets are usually discovered using *level-wise algorithms*, which divide the problem into multiple iterations of database scanning and counting occurrences of candidate itemsets of equal size.

Due to long execution times, data mining queries are often performed in a *batch mode*, where users submit sets of data mining queries to be executed during low

* This work was partially supported by the grant no. 4T11C01923 from the State Committee for Scientific Research (KBN), Poland.

database activity time (e.g., night time). It is likely that the batches contain data mining queries that operate on similar parts of the database. If such queries are executed separately, the same parts of the database are retrieved multiple times. We could reduce the overall I/O activity of the batch of data mining queries if we integrated their data retrieval operations on the same portions of the database.

For a system with unlimited memory, the integration of execution of multiple data mining queries consists in *common counting* [13][14] of candidate itemsets for all the queries so that every portion of the database needs to be read only once per iteration. However, if the memory is limited, we are not able to keep all candidate itemsets of all the data mining queries in the memory at the same time. The whole process must then be split into multiple phases of loading and counting the candidates, and therefore the data mining queries must be divided into subsets to be executed in each phase. We refer to the problem of dividing the data mining into subsets as to the *data mining query scheduling*.

In this paper we discuss the problem of data mining query scheduling and we introduce a heuristic algorithm to perform the scheduling for a system with limited memory. The goal of the algorithm is to schedule the data mining queries in such a way that the overall I/O cost for the whole batch is minimized.

1.1. Related Work

The problem of mining association rules was first introduced in [1] and an algorithm called *AIS* was proposed. In [2], two new algorithms were presented, called *Apriori* and *AprioriTid* that are fundamentally different from the previous ones. The algorithms achieved significant improvements over *AIS* and became the core of many new algorithms for mining association rules. *Apriori* and its variants first generate all frequent itemsets (sets of items appearing together in a number of database records meeting the user-specified support threshold) and then use them to generate rules. *Apriori* and its variants rely on the property that an itemset can only be frequent if all of its subsets are frequent. It leads to a level-wise procedure. First, all possible 1-itemsets (itemsets containing 1 item) are counted in the database to determine *frequent 1-itemsets*. Then, frequent 1-itemsets are combined to form potentially frequent 2-itemsets, called *candidate 2-itemsets*. Candidate 2-itemsets are counted in the database to determine *frequent 2-itemsets*. The procedure is continued by combining the frequent 2-itemsets to form *candidate 3-itemsets* and so forth. A disadvantage of the algorithm is that it requires K or $K+1$ passes over the database to discover all frequent itemsets, where K is the size of the greatest frequent itemset found.

In [4], an algorithm called *FUP* (Fast Update Algorithm) was proposed for finding the frequent itemsets in the expanded database using the old frequent itemsets. The major idea of *FUP* algorithm is to reuse the information of the old frequent itemsets and to integrate the support information of the new frequent itemsets in order to reduce the pool of candidate itemsets to be re-examined. Another approach to incremental mining of frequent itemsets was presented in [11]. The algorithm introduced there required only one database pass and was applicable not only for expanded but also for reduced database. Along with the itemsets, a *negative border* [12] was maintained.

In [10] the issue of interactive mining of association rules was addressed and the concept of *knowledge cache* was introduced. The cache was designed to hold frequent itemsets that were discovered while processing other queries. Several cache management schemas were proposed and their integration with the *Apriori* algorithm was analyzed. An important contribution was an algorithm that used itemsets discovered for higher support thresholds in the discovery process for the same task, but with a lower support threshold.

The notion of data mining queries (or *KDD* queries) was introduced in [6]. The need for Knowledge and Data Management Systems (KDDMS) as second-generation data mining tools was expressed. The ideas of application programming interfaces and data mining query optimizers were also mentioned. Several data mining query languages that are extensions of *SQL* were proposed [3][5][7][8][9].

2. Basic Definitions and Problem Formulation

Definition. Frequent itemsets.

Let $L=\{l_1, l_2, ..., l_m\}$ be a set of literals, called items. Let a non-empty set of items T be called an *itemset*. Let D be a set of variable length itemsets, where each itemset $T \subseteq L$. We say that an itemset T *supports* an item $x \in L$ if x is in T. We say that an itemset T *supports* an itemset $X \subseteq L$ if T supports every item in the set X. The *support* of the itemset X is the percentage of T in D that support X. The problem of mining frequent itemsets in D consists in discovering all itemsets whose support is above a user-defined support threshold.

Definition. Apriori algorithm.

Apriori is an example of a level-wise algorithm for association discovery. It makes multiple passes over the input data to determine all frequent itemsets. Let L_k denote the set of frequent itemsets of size k and let C_k denote the set of candidate itemsets of size k. Before making the k-th pass, *Apriori* generates C_k using L_{k-1}. Its candidate generation process ensures that all subsets of size $k-1$ of C_k are all members of the set L_{k-1}. In the k-th pass, it then counts the support for all the itemsets in C_k. At the end of the pass all itemsets in C_k with a support greater than or equal to the minimum support form the set of frequent itemsets L_k. Figure 1 provides the pseudocode for the general level-wise algorithm, and its *Apriori* implementation. The *subset(t, k)* function gives all the subsets of size k in the set t.

This method of pruning the C_k set using L_{k-1} results in a much more efficient support counting phase for *Apriori* when compared to the earlier algorithms. In addition, the usage of a hash-tree data structure for storing the candidates provides a very efficient support-counting process.

C_1 = {all 1-itemsets from D};
for (k=1; $C_k \neq \emptyset$; k++) **do**
begin
  count(C_k, D);
  $L_k = \{c \in C_k \mid c.count \geq minsup\}$;
  C_{k+1} = generate_candidates(L_k);
end;
Answer = $\bigcup_k L_k$;

L_1 = {frequent 1-itemsets};
for (k = 2; $L_{k-1} \neq \emptyset$; k++) **do**
begin
  C_k = generate_candidates(L_{k-1});
  forall tuples $t \in D$ **do**
  begin
    $C_t = C_k \cap$ subset(t, k);
    forall candidates $c \in C_t$ **do**
      $c.count$++;
  end;
  $L_k = \{c \in C_k \mid c.count \geq minsup\}$
end;
Answer = $\bigcup_k L_k$;

Figure 1. A general level-wise algorithm for association discovery (left) and its Apriori implementation (right).

Definition. Data mining query.

A *data mining query* is a tuple (R, a, Σ, Φ, β), where R is a database relation, a is an attribute of R, Σ is a selection predicate on R, Φ is a selection predicate on frequent itemsets, β is the minimum support for the frequent itemsets.

Example. Given is the database relation $R_1(attr_1, attr_2)$. The data mining query $dmq_1 = (R_1, "attr_2", "attr_1 > 5", "|itemset|<4", 3)$ describes the problem of discovering frequent itemsets in the set-valued attribute $attr_2$ of the relation R_1. The frequent itemsets with support above 3 and length less than 4 are discovered in records having $attr_1 > 5$.

Definition. Multiple data mining query optimization.

Given is a set of data mining queries $DMQ = \{dmq_1, dmq_2, ..., dmq_n\}$, where $dmq_i = (R, a, \Sigma_i, \Phi_i, \beta_i)$, Σ_i is of the form "$(l^i_{1min} < a < l^i_{1max}) \vee (l^i_{2min} < a < l^i_{2max}) \vee .. \vee (l^i_{kmin} < a < l^i_{kmax})$", and there are at least two data mining queries $dmq_i = (R_i, a, \Sigma_i, \Phi_i, \beta_i)$ and $dmq_j = (R, a, \Sigma_j, \Phi_j, \beta_j)$ such that $\sigma_{\Sigma i}R \cap \sigma_{\Sigma j}R \neq \emptyset$. The problem of *multiple data mining query optimization* is to generate an algorithm to execute DMQ with the minimal I/O cost.

Definition. Data sharing graph.

Let $S = \{s_1, s_2, ..., s_k\}$ be a *set of elementary data selection predicates* for DMQ, i.e., selection predicates over the attribute a or the relation R such that for all i,j we have $\sigma_{si}R \cap \sigma_{sj}R = \emptyset$ and for each i there exist integers $a, b, ..., m$ such that $\sigma_{\Sigma i}R = \sigma_{sa}R \cup \sigma_{sb}R \cup .. \cup \sigma_{sm}R$ (example in Fig. 2). A graph $DSG=(V,E)$ is called a *data sharing graph* for the set of data mining queries DMQ iff $V = DMQ \cup S$, $E = \{(dmq_i, s_j) \mid dmq_j \in DMQ, s_j \in S, \sigma_{\Sigma i}R \cap \sigma_{sj}R \neq \emptyset\}$.

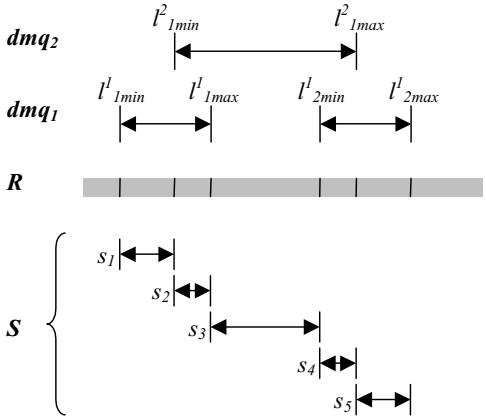


Figure 2. Example set of data mining queries and their elementary data selection predicates.

Example. Given is the relation $R_1=(attr_1, attr_2)$ and three data mining queries: $dmq_1=(R_1, "attr_2", "5 < attr1 < 20", \varnothing, 3)$, $dmq_2=(R_1, "attr_2", "10 < attr1 < 30", \varnothing, 5)$, $dmq_3=(R_1, "attr_2", "15 < attr1 <40", \varnothing, 4)$. The set of elementary data selection predicates is then $S=\{s_1="5<attr_1<10", s_2="10<attr_1<15", s_3="15<attr_1<20", s_4="20<attr_1<30", s_5="30<attr_1<40"\}$. The data sharing graph for $\{dmq_1, dmq_2, dmq_3\}$ is shown in Fig. 3.

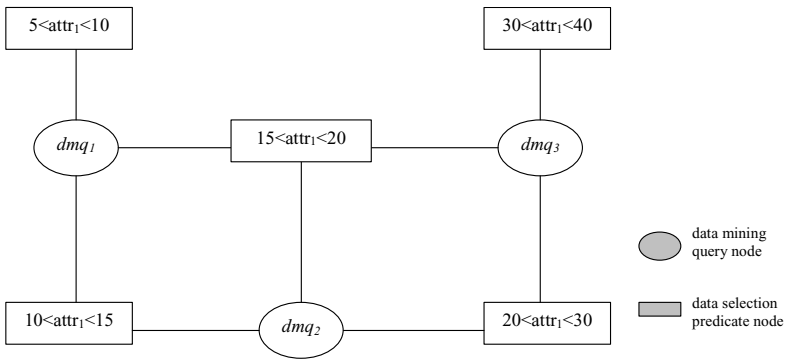


Figure 3. Example data sharing graph.

Definition. Apriori Common Counting.

A straightforward way to perform multiple data mining query optimization is *Apriori Common Counting* algorithm. The algorithm proceeds as follows. In the first step, *Apriori Common Counting* constructs separate candidate 1-itemset hash trees (in memory) for each data mining query. Next, all database partitions corresponding to the elementary selection predicates are scanned and the candidate itemsets for the appropriate data mining queries are counted. This process is repeated for each iteration: for candidate 2-itemsets, candidate 3-itemsets, etc. Notice that if a given elementary selection predicate is shared by multiple data mining queries, then the

specific part of the database needs to be read only once (per iteration). This property helps reduce the overall I/O cost of batched data mining query execution. The idea of *Apriori Common Counting* algorithms is depicted in Fig. 4.

```
for (i=1; i<=n; i++)                    /* n = number of data mining queries */
  C_1^i = {all 1-itemsets from σ_{s1∪s2∪..∪sk}R, ∀s_j∈S: (dmq_i,s_j)∈E}  /* generate 1-candidates */
for (k=1; C_k^1 ∪ C_k^2 ∪..∪ C_k^n ≠ ∅; k++) do begin
  for each s_j∈S do begin
    CC= ∂C_k^l: (dmq_l,s_j)∈E;          /* select the candidates to count now */
    if CC≠ ∅ then count(CC, σ_{sj}R);
  end;
  for (i=1; i<=n; i++) do begin
    L_k^i = {c ∈ C_k^i | c.count ≥ minsup^i};          /* identify frequent itemsets */
    C_{k+1}^i = generate_candidates(L_k^i);
  end;
end;
for (i=1; i<=n; i++) do
  Answer^i = ∪_k L_k^i;                 /* generate responses */
```

Figure 4. Apriori Common Counting.

3. Data Mining Query Scheduling

The basic *Apriori Common Counting* described in the previous section assumes unlimited memory for its operation. However, if the memory is limited, then it is not possible to construct candidate hash trees for all the data mining queries. The whole algorithm must then be split into multiple *phases* and every phase must consist in executing a subset of the data mining queries. The key problem is which data mining queries should be performed in the same phase and which of them can be performed in separate phases. The task of dividing the set of data mining queries into subsets is referred to as *data mining query scheduling*.

There are several aspects to consider when designing a data mining query scheduling algorithm. Firstly, it is obvious that system memory size restricts the number of data mining queries that may be processed in the same phase. Memory requirements for the data mining queries are based on sizes of their candidate hash trees, which in turn depend on data characteristics and the specific iteration of the algorithm (typically, sizes of candidate hash trees systematically reduce for iterations 3, 4, etc.). Since the candidate hash tree sizes change in each iteration, the data mining query scheduling algorithm should be used before generating every new tree, not only at the beginning of the data mining query processing. Another aspect is that the goal of *Apriori Common Counting* is to reduce the overall I/O activity. Therefore, similarities between data mining queries should be taken into account when putting data mining queries into the same phase. Data mining queries that operate on separate portions of the database can be processed in separate phases, while data mining queries that operate on highly overlapping database portions should be executed in the same phase. To measure the “overlapping” between data

mining queries one can rely on a traditional DBMS query optimizer, which estimates predicate costs based on database statistics.

In order to schedule data mining queries, the sizes of their candidate hash trees must be known. There are two options to derive the size. The first option is to calculate the upper bounds on the candidate hash trees and use the upper bounds in the scheduling algorithm. The upper bounds can be evaluated based on the number of frequent itemsets discovered in the previous iteration. A disadvantage of this approach is that the real candidate hash trees are smaller than the estimates, so the scheduling algorithm is likely to miss the optimal solution. The second option is to generate the candidate hash trees first, measure their sizes, save them in temporary files, perform the scheduling and then retrieve the appropriate trees from the files while performing the phases. The main advantage of this approach is that the scheduling algorithm operates on the exact sizes of the trees, and therefore it is able to find the optimal solution. However, the additional I/O cost is introduced because of the need to temporarily store the candidate hash trees on disk. Nevertheless, when dealing with very large databases (in case of which candidate tree sizes are by several orders of magnitude smaller than the database) that extra cost is going to be compensated by reduction of database reads thanks to *Common Counting*.

Let us consider an example of data mining query scheduling based on our previous set of data mining queries from Fig. 3. Let *cost(s)* be the I/O cost of retrieving database records that satisfy the data selection predicate *s*. Let *treesize(dmq,k)* be the *k*-item candidate hash tree size for the data mining query *dmq*. Sample costs and tree sizes (e.g., for the third *Apriori* iteration) are given in the table below. Let us assume the system memory limit of 10MB, meaning that at most two of the data mining queries can fit in at a time (i.e., in one phase).

s_i	$cost(s_i)$
$5<attr_1<10$	5000
$10<attr_1<15$	7000
$15<attr_1<20$	2000
$20<attr_1<30$	2000
$30<attr_1<40$	1000

dmq_i	$treesize(dmq_i,3)$
dmq_1	4M
dmq_2	5M
dmq_3	3M

There exist four different schedules that satisfy the given constraints. The schedules and the total costs of executing the sample set of data mining queries are given below. The *Schedule A* represents a sequential execution of all the data mining queries. One can notice that the optimal solution is the *Schedule B*, which reduces the overall cost by 30%. This schedule has been also depicted in Fig. 5.

Schedule A

phase	*data mining queries*	*trees size*	*data selection predicates*	*phase cost*
1	dmq_1	4M	s_1, s_2, s_3	14,000
2	dmq_2	5M	s_2, s_3, s_4	11,000
3	dmq_3	3M	s_3, s_4, s_5	5,000
total cost				**30,000**

Schedule B

phase	*data mining queries*	*trees size*	*data selection predicates*	*phase cost*
1	dmq_1, dmq_2	9M	s_1, s_2, s_3, s_4	16,000
2	dmq_3	3M	s_3, s_4, s_5	5,000
total cost				**21,000**

Schedule C

phase	*data mining queries*	*trees size*	*data selection predicates*	*phase cost*
1	dmq_1, dmq_3	7M	s_1, s_2, s_3, s_4, s_5	17,000
2	dmq_2	5M	s_2, s_3, s_4	11,000
total cost				**28,000**

Schedule D

phase	*data mining queries*	*trees size*	*data selection predicates*	*phase cost*
1	dmq_2, dmq_3	8M	s_2, s_3, s_4, s_5	12,000
2	dmq_1	4M	s_1, s_2, s_3	14,000
total cost				**26,000**

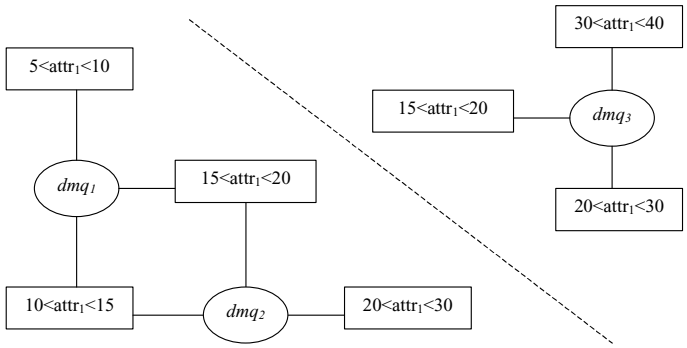


Figure 5. The optimal schedule for the sample set of data mining queries.

The data mining query scheduling problem can be solved using a combinatorial approach, in which all possible (allowable) schedules are generated first, and then their overall costs are calculated. The combinatorial approach can be suitable for a small number of data mining queries in the set, however, for complex problems, involving large numbers of data mining queries, the overhead of the approach would be unacceptable. For a given number of data mining queries, the number of all possible schedules is determined by Bell number – e.g., for 13 queries the number of schedules exceeds 4 millions. Therefore we introduce a heuristic algorithm for finding suboptimal schedules for executing a set of data mining queries.

3.1. Heuristic Scheduling Algorithm: CCRecursive

The algorithm iterates over all the elementary selection predicates, sorted in descending order with respect to their I/O costs. For each elementary selection predicate we identify all the data mining queries that include the predicate. If none of the identified queries has been already scheduled, then we create a new phase and we put all the queries into the new phase. Otherwise, we merge the phases to which the scheduled queries belonged and we assign the other queries to this new phase. If the size of the newly created phase exceeds the memory limit, then the phase is split into smaller ones by recursive execution of the algorithm. At the end of the algorithm, we perform *phase compression*, which consists in merging those phases that do not consume all the available memory. The detailed structure of the algorithm is given in Fig. 6. The auxiliary function *treesize(Q)*, where Q is a set of data mining queries, represents total memory size required to hold candidate hash trees for all the data mining queries in Q.

```
Phases ← {∅}
sort S = <s_i , s_2 ,..., s_k> in descending order with respect to cost(s_i)
CCRecursive(S, DMQ, Phases):
begin
  ignore in S those predicates that are used by less than two dmqs;
  for each s_i in S do begin
    tmpDMQ ← { dmq_j | dmq_j =(R, a, Σ_j, Φ_j, β_j), s_i ⊆Σ_j , dmq_j ∈ DMQ};
    commonPhases ← {p ∈ Phases | p∩ tmpDMQ ≠∅};
    if commonPhases = ∅ then
      newPhase ← tmpDMQ;
    else
      newPhase ← tmpDMQ ∪ U p| p∈ commonPhases;
    end if;
    if treesize(newPhase) ≤MEMSIZE then
              Phases ← Phases \ commonPhases;
              Phases ← Phases ∪ newPhase;
    else
          Phases ← CCRecursive(<s_i+1, … s_k>, newPhase, Phases);
    end if;
  end;
  add phase for each unscheduled query;
  compress Phases containing queries from DMQ;
  return Phases;
end.
```

Figure 6. Heuristic scheduling algorithm: CCRecursive.

4. Experimental Evaluation

To evaluate our heuristic algorithm *CCRecursive* we performed a series of simulations on a PC with *AMD Duron 1200 MHz* processor and 256 MB of main memory. We focused on the isolated problem of scheduling queries into phases fitting in main memory in a given iteration of *Common Counting*. We compared the amount of data read from the database by our heuristic algorithm and the complete "brute-force" algorithm testing all possible assignments of queries to phases.

We simulated actual batches of frequent set discovery tasks by randomly generating a collection of queries. For each query, the database selection predicate and the size of candidate tree was randomly generated. Then the amount of total main memory was also randomly chosen in such a way that the number of queries fitting into it ranged from one query to all the queries.

We performed several series of experiments varying the number of queries. Each of the series consisted of 100 simulations. Figure 7 presents how the accuracy of our heuristic algorithm changes with the number of queries. To assess the accuracy we measured the relative amount of data read from the database by schedules generated by our heuristics compared to the optimal schedules (generated by the complete brute-force scheduling algorithm). For example, in the case of 11 queries, *CCRecursive* generates schedules that read on average about 3.5% more data than the optimal schedules.

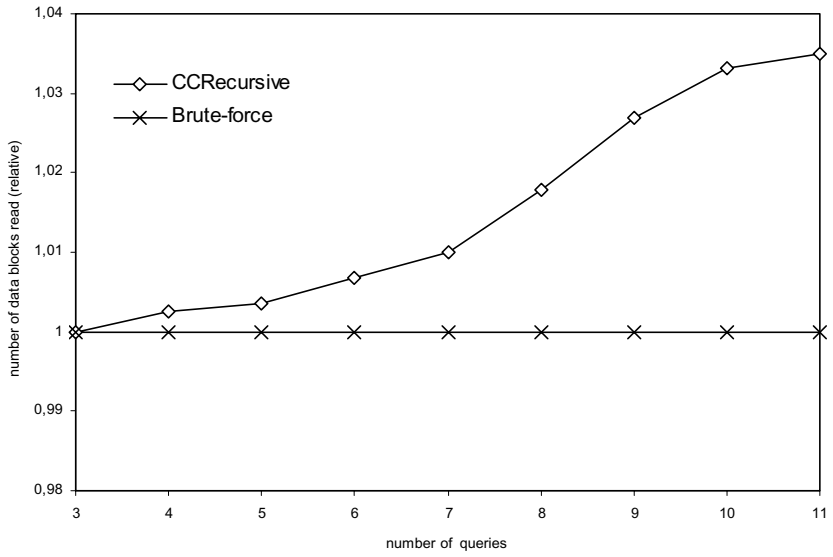


Figure 7. Amounts of data read by CCRecursive schedules and optimal schedules.

Figure 8 presents the execution times (times needed to generate schedules) of *CCRecursive* and the brute-force algorithm. Although *CCRecursive* still scales exponentially with the number of queries, its execution time increases less rapidly

than in case of the brute-force solution. For instance, the brute-force algorithm consumes more than 1000 s already for 12 queries, while *CCRecursive* exceeds that threshold in case of 22 queries (the chart presents the times for up to 15 queries).

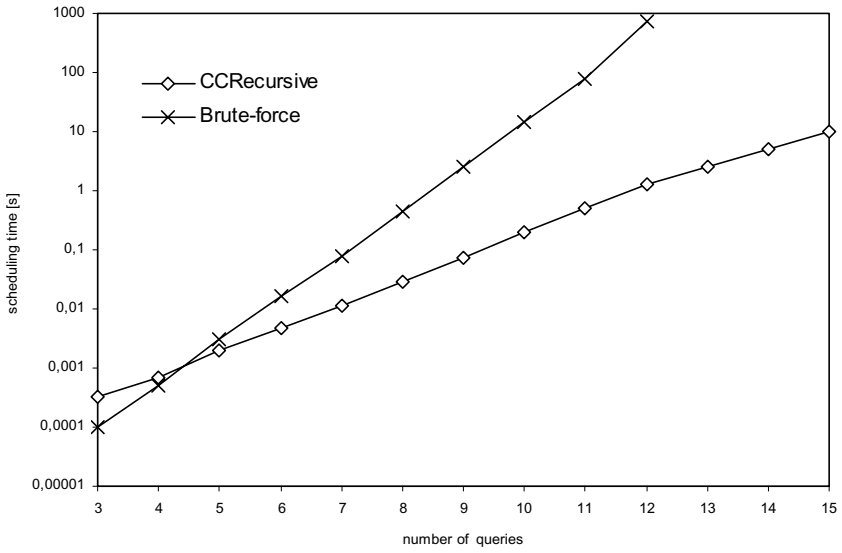


Figure 8. Execution times (logarithmic scale) of CCRecursive and the brute-force scheduling algorithm.

The results of conducted experiments show that *CCRecursive* significantly outperforms the brute-force solution (with the exception of cases with 3 and 4 queries when execution times of both algorithms are negligible), which makes it applicable for larger batches of data mining queries. We believe that the accuracy of our heuristics (shown in Fig. 7) is acceptable. However, it should be noted that the actual trade-off between extra disk accesses (introduced by the heuristics) and reduction in the scheduling time cannot be assessed without knowing the database size and hardware parameters.

5. Concluding Remarks

In this paper we addressed the problem of *common counting* of candidate itemsets for multiple data mining queries. We have formally defined the problem of *data mining query scheduling*, which consists in splitting the set of data mining queries into subsets (phases) such that the candidate hash trees can fit in limited memory and the overall I/O cost is minimized.

Since the number of possible schedules growth rapidly with the number of queries, we proposed a heuristic scheduling algorithm, called *CCRecursive*. The experiments show that our heuristics generates schedules that are close to optimal and is more efficient than the brute-force solution and thus applicable for much greater number of queries.

References

[1] Agrawal R., Imielinski T., Swami A. Mining Association Rules Between Sets of Items in Large Databases. Proc. of the 1993 ACM SIGMOD Conf. on Management of Data, 1993.
[2] Agrawal R., Srikant R. Fast Algorithms for Mining Association Rules. Proc. of the 20th Int'l Conf. on Very Large Data Bases, 1994.
[3] Ceri S., Meo R., Psaila G. A New SQL-like Operator for Mining Association Rules. Proc. of the 22nd Int'l Conference on Very Large Data Bases, 1996.
[4] Cheung D.W., Han J., Ng V., Wong C.Y. Maintenance of Discovered Association Rules in Large Databases: An Incremental Updating Technique. Proc. of the 12th ICDE, 1996.
[5] Han J., Fu Y., Wang W., Chiang J., Gong W., Koperski K., Li D., Lu Y., Rajan A., Stefanovic N., Xia B., Zaiane O.R. DBMiner: A System for Mining Knowledge in Large Relational Databases. Proc. of the 2nd KDD Conference, 1996.
[6] Imielinski T., Mannila H. A Database Perspective on Knowledge Discovery. Communications of the ACM, Vol. 39, No. 11, 1996.
[7] Imielinski T., Virmani A., Abdulghani A. Datamine: Application programming interface and query language for data mining. Proc. of the 2nd KDD Conference, 1996.
[8] Morzy T., Wojciechowski M., Zakrzewicz M. Data Mining Support in Database Management Systems. Proc. of the 2nd DaWaK Conference, 2000.
[9] Morzy T., Zakrzewicz M. SQL-like Language for Database Mining. ADBIS'97 Symposium, 1997.
[10] Nag B., Deshpande P.M., DeWitt D.J. Using a Knowledge Cache for Interactive Discovery of Association Rules. Proc. of the 5th KDD Conference, 1999.
[11] Thomas S., Bodagala S., Alsabti K., Ranka S. An Efficient Algorithm for the Incremental Updation of Association Rules in Large Databases. Proc. of the 3rd KDD Conference, 1997.
[12] Toivonen H. Sampling Large Databases for Association Rules. Proc. of the 22nd Int'l Conference on Very Large Data Bases, 1996.
[13] Wojciechowski M., Zakrzewicz M. Methods for Batch Processing of Data Mining Queries. Proc. of the 5th International Baltic Conference on Databases and Information Systems, 2002.
[14] Wojciechowski M., Zakrzewicz M. Evaluation of Common Counting Method for Concurrent Data Mining Queries. Proc. of the 7th ADBIS Conference, 2003.

SCIENTIFIC PAPERS UNIVERSITY OF LATVIA. 2004. VOLUME 672, Datbases and Information Systems
282–290 P.

OpAC: A New OLAP Operator Based on a Data Mining Method

Riadh Ben Messaoud [*], Sabine Rabaséda [**], Omar Boussaid [**], Fadila Bentayeb [*]

Laboratoire ERIC – Université Lumière Lyon 2
5 avenue Pierre Mendès-France
69676 Bron Cedex – France
http://eric.univ.lyon2.fr
[*]{rbenmessaoud, bentayeb}@eric.univ-lyon2.fr
[**]{sabine.rabaseda, boussaid}@univ-lyon2.fr

Abstract. For a few years, on-line analysis processing (OLAP) and data mining have known parallel and independent evolutions. Some recent studies have shown the interest of the association of these two fields. Currently, we attend the increase of a more elaborated analysis's need. We think that the idea of coupling OLAP and data mining will be able to fulfill this need. We propose to adopt this coupling in order to create a new operator, *OpAC* (Operator for Aggregation by Clustering), for multidimensional on-line analysis. The main idea of *OpAC* consists in using the agglomerative hierarchical clustering to achieve a semantic aggregation on the attributes of a data cube dimension.

Keywords. On-line analysis processing, Data cubes, Data mining, Agglomerative hierarchical clustering, Semantic aggregation.

1. Introduction

Data warehouses provided several solutions to the management of huge amount of data [9]. In fact, a data warehouse is an analysis oriented structure that stores a large collection of subject-oriented, integrated, time variant and non-volatile data. The warehousing process starts by extracting, transforming and loading data from heterogeneous sources (ETL). Some particular models, such as the star schema and the snow-flaked schema, are designed in order to prepare integrated data to analysis using the on-line analytical processing technology (OLAP). These models support decision making tasks by exploring multidimensional data views, commonly called *data cubes* [1]. So far, a data warehouse becomes a large infrastructure for designing efficient decision process through visualization and navigation into large data volumes.

On the other side, data mining uses machine learning methods to discover, describe and predict non trivial patterns from data. These patterns are usually expressed in valid and understandable models. However, data mining is a dependent step in the process of knowledge discovery in databases. In fact, all data mining methods need to work on integrated, consistent and cleaned data, which often requires data cleaning as preprocessing steps [5].

OLAP and data mining have known parallel and independent evolutions. For long, they were considered as two different fields. Currently, we think that their association could allow a more elaborated OLAP task exceeding the simple exploration of a data cube.

In one hand, OLAP is characterized by its aggregation tools, its navigational aspect and its power for visualizing data. In the other hand, data mining is known for the descriptive and predictive power of its results. Moreover, we think that multidimensional data structure can provide a suitable context for applying data mining methods. Our purpose is to take advantage as well from OLAP as from data mining and to integrate them in the same analysis process to provide exploration, explication and prediction capabilities. We look, particularly, for improving the traditional OLAP operators by creating a new form of aggregation based on a data mining method. Taking into account the multidimensional structure of data and the need to integrate them in a more elaborated analysis process, our idea consists in developing a new aggregation operator, called *OpAC* (Operator for Aggregation by Clustering), and based on the AHC (Agglomerative Hierarchical Clustering) [10].

The remaining of this paper is organized as follows. In section 2, we expose the related works to the coupling between OLAP and data mining. In section 3, we present the objectives of our proposed operator. In section 4, we motivate why we choose the AHC as an aggregation method. We develop, in section 5, a theoretical formalization for the *OpAC* operator. In section 6, we propose an implementation of a prototype and in section 7, we conclude our work and propose some future research topics.

2. Related work

A few research studies deal with the coupling of OLAP and data mining. This is due partly to the fact that most of the attention is directed towards separated improvements of the two areas. Nevertheless, we distinguish three principal groups of approaches:

The first approach consists in simulating the data mining methods by extending OLAP operators. Han proposes a system, called *DBMiner*, which can perform some data mining functions including association, classification, prediction, clustering, and sequencing [8]. Chen et al. suggest an approach consisting in mining functional association rules using the distributed OLAP and data mining infrastructure [3]. The purpose of this work is to enhance the expressive power of association rules. The infrastructure mines e-commerce transaction data and generate association rules expressing customer behavior patterns. Goil and Choudhary propose to mine knowledge from data cubes by using the OLAP operators [6]. Their approach consists in discovering association rules from the quantitative summary information contained in a data cube.

The second approach aims to adapt multidimensional data in order to make them understandable by data mining methods. Two strategies are proposed.

One consists in taking advantages from multidimensional database management system (MDBMS) to help the construction of learning models. For instance, Laurent proposes a cooperation between *Oracle Express* and a fuzzy decision tree software (Salammbô) [11]. This cooperation allows transferring learning tasks, storage constraints and data handling to the MDBMS.

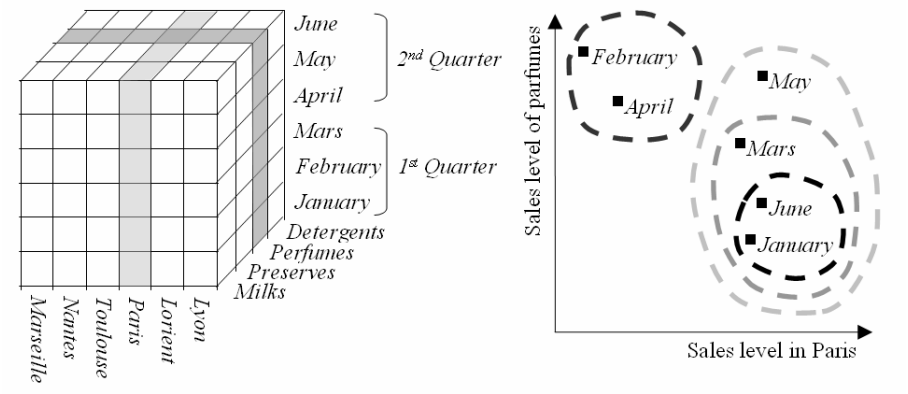
Another strategy transforms the multidimensional data and makes them usable by the data mining methods. Pinto et al. integrate multidimensional information in data sequences and apply on them the discovery of frequent patterns [12]. In order to implement a decision tree on multidimensional data, Goil and Choudhary flatten data cubes to extract contingency matrix for each dimension at each construction step of the tree [7]. Chen et al. propose to adopt OLAP as a preprocessing step of the knowledge discovery process [2]. Transformed data can therefore be exploited by data mining methods.

The third approach aims to adapt data mining algorithms and employs them directly in multidimensional data. Sarawagi et al. propose to integrate a statistical module, based on multidimensional regression, (*Discovery-driven*) in OLAP server. This module guides the user to detect relevant areas at various hierarchical levels of a cube [13]. In [14], Sarawagi proposes a new tool, *iDiff*, based on dynamic programming, which detects both relevant areas in a data cube and the reasons of their presence. Similar works were released in [4] for the generation of natural language from multidimensional data.

Finally, we note that none of the existing approaches employs the coupling between data mining and OLAP in order to enhance the functionalities of OLAP operators. Our approach associates the exploratory tools of OLAP with the descriptive and predictive aspect of data mining. The current work aims to define a new generation of analysis operators based on data mining methods. We propose in this paper a new operator based on a data mining method to fulfill more elaborated analysis.

3. *OpAC* operator objectives

The construction of a data cube targets precise analysis goals. The selection of its dimensions and measures depends on the analysis needs. Usually, a dimension is organized according several hierarchies expressing various levels of granularity. Each hierarchy contains a set of modalities, and each modality of a hierarchy includes modalities from the hierarchy immediately below according to the logical membership order.



(a) OLAP context (b) Clustering context

Figure 1. Principle of the aggregation operator *OpAC*.

In fact, the modalities of a dimension are always organized according to the logical order of membership well known in the natural use of objects and concepts of the real world. Let's consider the data cube presented in Fig.1(a). The cube is made up of three dimensions: *Location*, *Time* and *Product*. The *Time* dimension is organized according to two hierarchical levels: *Months* and *Quarters*. It is natural to say that the modality "*1st Quarter*" of the temporal dimension aggregates the months: "*January*", "*February*" and "*March*".

Unlike the traditional OLAP aggregation, exposed above, our approach takes the cube measures into account in order to provide a *semantic* aggregation over dimension modalities. The goal of our operator *OpAC* is to use a clustering method in order to highlight aggregates semantically richer than those provided by the current OLAP operators.

As shown in Fig.1(b), the new operator enables us to note that "*January*" and "*June*" form a more significant aggregate since they represent periods where sales level of "*Perfumes*" in the city of "*Paris*" are slightly similar.

Existing OLAP tools, like the *Slicing* operator, can also create new modalities' aggregates in a cube dimension. Therefore, these tools always need handmade user assistance, whereas our operator is based on a clustering algorithm that provides automatically relevant aggregates. Furthermore, with classical OLAP tools, aggregates are created in an intuitive way in order to compare some measure values, whereas *OpAC* creates significant aggregates expressing deep relations with the cube's measures. Thus, the construction of this kind of aggregates is very interesting to establish a richer on-line analysis context.

4. The choice of the agglomerative hierarchical clustering

According to our *OpAC* operator objectives, we chose the agglomerative hierarchical clustering (AHC) as an aggregation method for the *OpAC* operator. This choice is motivated by the following points:

- The hierarchical aspect constitutes a relevant analogy between the AHC results and a hierarchical structure of a dimension. Furthermore, the objectives and the results representation expected for *OpAC* match perfectly with the AHC strategy;
- Unlike the DHC (Divisive Hierarchical Clustering), the AHC adopts an agglomerative strategy beginning by the finest partition where each individual is considered like a class. This allows including the finest modalities of a dimension in the results of *OpAC*. Moreover, the ascending strategy is faster than the divisive one;
- The results of the AHC are compatible with exploratory aspect of OLAP and can be reused by its classical operators. The AHC provides several hierarchical partitions of individuals. By moving from a partition level to the higher one, two aggregates are joined together. Conversely, by moving from a partition level to the lower one, an aggregate is divided into two new aggregates. These operations are strongly similar to the classical operators *Roll-up* and *Drill-down*.

5. *OpAC* operator formalization

This formalization defines the individuals and the variables domains for the clustering problem. Let's Ω be the set of individuals and Σ the set of variables. We suppose that:

- C is a data cube having d dimensions and m measures;
- $D_1,\ldots,D_i,\ldots,D_d$ the dimensions of C;
- $M_1,\ldots,M_q,\ldots,M_m$ the dimensions of C;
- $\forall i \in \{1,\ldots,d\}$ the dimension D_i contains n_i hierarchical levels;
- h_{ij} the j^{th} hierarchical level of D_i, where $j \in \{1,\ldots,n_i\}$;
- $\forall j \in \{1,\ldots,n_i\}$ the hierarchical level h_{ij} contains l_{ij} modalities;
- g_{ijt} the t^{th} modality of h_{ij}, where $t \in \{1,\ldots,l_{ij}\}$;
- $G(h_{ij})$ the set of modalities of h_{ij}.

Let's consider the modalities of h_{ij} as the set of individuals. i.e.

$$\Omega = G(h_{ij}) = \{g_{ij1}, \ldots, g_{ijt}, \cdots, g_{ijl_{ij}}\}$$

We adopt now the following notations:

- $*$ a meta-symbol indicating the total aggregate of a dimension;
- $\forall q \in \{1,\ldots,m\}$ we define the measure M_q as the function: $M_q : G \to \Re$;
- G the set of d-tuples of all the hierarchies modalities of the cube C including the total aggregates of dimensions;

$$G = \prod_{i=1}^{d} \left(G(h_{ij}) \cup \{*\} \right)_{j \in \{1,\ldots,n_i\}}$$
$$= \left(G(h_{1j}) \cup \{*\} \right)_{j \in \{1,\ldots,n_1\}} \times \cdots \times \left(G(h_{ij}) \cup \{*\} \right)_{j \in \{1,\ldots,n_i\}} \times \cdots \times \left(G(h_{dj}) \cup \{*\} \right)_{j \in \{1,\ldots,n_d\}}$$

Reconsider again the cube of Fig.1(a), with the dimensions: D_1 (*Time*), D_2 (*Location*), D_3 (*Product*) and the measure M_1 (*Sales level*). For instance, $M_1(\textit{February 1999}, \textit{Lyon}, *)$ indicates the sales level of all products in *February 1999* for the city of *Lyon*.

We adopt the cube measures as quantitative variables describing the population $\Omega = G(h_{ij})$. Nevertheless, in order to insure their statistical and logical validity, it is necessary to respect two fundamental constraints in the choice of these variables.

- **First constraint:** Hierarchical levels belonging to the dimension D_i, retained for the individuals, can not generate variables. In fact, describing an individual by a property which contains it has no logical sense. Conversely, a variable which specifies a property of an individual can only serve for the description of this particular individual;
- **Second constraint:** In order to insure the independence of variables, by dimension, only one hierarchical level can be chosen to generate them. In fact, the value taken by a modality can be obtained by linear combination of modalities belonging to the lower hierarchy.

Therefore, all possible extracted variables belong to the following set:

$$\Sigma \subset \left\{ \begin{array}{l} X \,/\, \forall t \in \{1,\ldots,l_{ij}\} \\ X(g_{ijt}) = M_q(*,\ldots,*, \underbrace{g_{ijt}}_{j \in \{1,\ldots,n_j\}}, *,\ldots,*, \underbrace{g_{srv}}_{r \in \{1,\ldots,n_s\}}, *,\ldots,*) \\ \text{with } s \neq i,\ r \text{ is unique for each } s,\ v \in \{1,\ldots,l_{sr}\} \text{ and } q \in \{1,\ldots,m\} \end{array} \right\}$$

To enhance the understanding of this formalization, we reconsider the cube of Fig.1(a). Let's suppose that an expert wishes to classify months according to their sales levels by location and/or by product. For this, we retain the modalities of the *Months* level of the dimension D_1 as the set of individuals, i.e. $\Omega = \{\textit{January}, \textit{February}, \textit{Mars}, \textit{April}, \textit{May}, \textit{June}\}$. Thus, we can choose one hierarchical level of D_2 and/or D_3 as generator of variables. For instance, if we choose *Cities* level of D_2, we operate total aggregations (*Roll-up*) on the rest of the cube's dimensions except D_1, the dimension retained for individuals, i.e. we roll-up totally D_3. We obtain a contingency table expressing sales levels by cities at each

month. In the same way, we can generate variables from D_3 by operating a total aggregation on D_2 .

6. Implementation

To illustrate our method, we propose a prototype[1] for the *OpAC* operator. Its implementation was realized with *Visual Basic* under *Windows XP Professional*. The setup of *MS SQL Server* and *MSOLAP driver* is necessary for the running of the prototype. Three principal components constitute our prototype:

- **A parameter setting interface** to assist user in the selection of individuals and variables from a data cube with respect to the above defined constraints. It allows, also, the selection of the clustering's parameters;

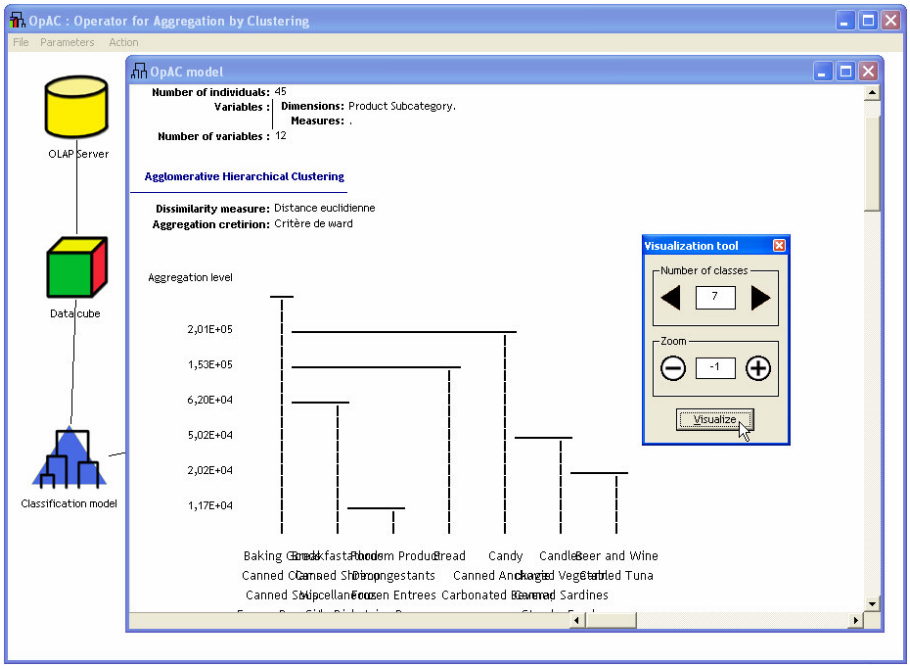


Figure 2. The *OpAC* prototype.

- **A data loading module** that ensures the connection to a data cube via the OLAP server; imports information about the cube's structure (labels of dimensions, hierarchies and measures); and loads data to be analyzed;

[1] http://bdd.univ-lyon2.fr/download/opac.zip

- **A clustering module** to construct the AHC model and plots its results via a dendrogram. The graphic representation of the dendrogram includes a summary of the AHC's parameters and the analyzed data.

As shown in Fig.2, we have provided our prototype with an interactive interface and several visual tools. These tools allow navigation into the dendrogram and a better interpretation of analyzed data.

7. Conclusion

The objective of our study is to satisfy the need of more elaborated on-line analysis. For this, we have created a new aggregation operator which integrates the AHC method into the multidimensional data structure. First, we identified the objectives we plan to *OpAC*. Then, we motivated the choice of AHC as a suited aggregation method. A theoretical formalization was proposed to define the individuals and variables of the clustering problem. We have validated our approach by implementing a prototype.

The *OpAC* operator distinguishes from classical OLAP operators by its ability to aggregate dimension modalities with respect to their semantic bounds. Its aggregates reflect real facts contained in a data cube. Our operator represents a possible way to realize elaborated on-line analysis. Moreover, our choice of the AHC does not exclude the use of other clustering methods. More generally, we think that the use of data mining methods would be suitable to establish new models of on-line learning on multidimensional data.

Finally, the *OpAC* operator can be enhanced in several possible ways. We plan to provide it with an evaluation tool to measure the quality of generated aggregates and to extend it in order to treat as well numerical as complex data cubes.

References

[1] Chaudhuri, S., Dayal, U. (1997) An Overview of Data Warehousing and OLAP Technology. SIGMOD Record. 26(1), 65 – 74.

[2] Chen, M., Zhu, Q.U., Chen, Z.X. (2001) An integrated interactive environment for knowledge discovery from heterogeneous data resources. Information and Software Technology. July 2001, 43(8), 487 – 496.

[3] Chen, Q., Dayal, U., Hsu, M. (2000) An OLAP-based Scalable Web Access Analysis Engine. In: 2nd International Conference on Data Warehousing and Knowledge Discovery (DAWAK'2000). September 2000, London, UK.

[4] Favero, E.L., Robin, J. (2001) Using OLAP and Data Mining for Content Planning in Natural Language Generation. Lecture Notes in Computer Science. 1959, 164 – 175.

[5] Fayyad, U.M., Shapiro, G.P., Smyth, P. et al. (1996) Advances in Knowledge Discovery and Data Mining. AAAI/MIT Press.

[6] Goil, S., Choudhary, A. (1998) High Performance Multidimensional Analysis and Data Mining. In: High Performance Networking and Computing Conference (SC'98). November 1998, Orlando, USA.

[7] Goil, S., Choudhary, A. (2001) PARSIMONY: An Infrastructure for parallel Multidimensional Analysis and Data Mining. Journal of parallel and distributed computing. 61(3), 285 – 321.
[8] Han, J. (1998) Toward On-line Analytical Mining in Large Databases. In: SIGMOD Record. 27(1), 97 – 107.
[9] Kimball, R. (1996) The Data Warehouse toolkit, John Wiley & Sons.
[10] Lance, G.N. and Williams, W.T. (1967) A general theory of clustering sorting strategies: Clustering systems. The Computer Journal. 10, 271 – 277.
[11] Laurent, A. (2001) De l'OLAP Mining au F-OLAP Mining. Revue Extraction des connaissances et apprentissage (ECA). Hermès (ed.), 1(1-2), 189 – 200.
[12] Pinto, H., Han, J., Dayal, U. et al. (2001) Multi-dimensional Sequential Pattern Mining. In: On Information and Knowledge Management (CIKM'01). November 2001, Atlanta, USA.
[13] Sarawgi, S., Agrawal, R., Megiddo, N. (1998) Discovery-driven Exploration of OLAP Data Cubes. In: Proceeding of the 6th Int'l Conference on Extending Database Technology (EDBT). Mars 1998, Valencia, Spain.
[14] Sarawagi, S. (2001) iDiff: Informative summarization of differences in multidimensional aggregates. Data Mining And Knowledge Discovery. 5(4), 213 – 246.

SCIENTIFIC PAPERS UNIVERSITY OF LATVIA. 2004. VOLUME 672, Datbases and Information Systems
291–305 P.

Controlling access to Data Warehouse data within the database

Laila Niedrite, Liga Grundmane

University of Latvia,Department of Computer Science
19 Raina boulevard, Riga, Latvia
lnied@lanet.lv, sd00099@lanet.lv

Abstract. There are some alternative approaches as to how to control user access to restricted information. The goal of this paper is to provide two models of data access control specific for data warehouses implemented in RDBMS. The models *"Role Based Security Model"* and "*Dynamically Adapting Views Based Data Access Control Model"* are based on the idea of revoking all information, restricting tasks from the application layer, which is important for data warehouses where different user tools may be provided. Our proposed two solutions also support the possibility for deriving data access rights from data sources.

Keywords. Data Warehouse, Data Access, Restriction

1. Introduction

The data warehouse is storage of data that is extracted and integrated from different data sources. A data warehouse often contains sensitive data and certain user/s are either prevented or permitted as the case may be from accessing unauthorized data as defined and set out in accordance to the various criteria's within either the *"Role Based Security Model"* or "*Dynamically Adapting Views Based Data Access Control Model"*. Both these models will be discussed later in this paper, however, as with any system or theoretical model/s, they must all be set up to serve their respective clients needs and over-all corporate objectives while maintaining system data integrity and security of certain types of data while maximizing the throughput time (such as query time) of the system itself. Both models offer inherent advantages and disadvantages and they must be considered if proper overall data base operation is a necessity, particularly in a setting where there are or could be literally thousands of users on line at one time. One of theses concerns is of particular importance and that relates to certain overlaps of users, which may through employment or other be required to perform multi functions which directly results in their ability to cross over access of certain data bases. The data access security system must be set up to recognize and distinguish and assign the proper access rights in such cases.

In particular how can consistent and simultaneous data access rights be accomplished and secured knowing that one person (as mentioned in the previous paragraph) may be a data source systems user with different data access rights to similar content data sets, e.g. PERSONS in one data source and EMPLOYEES of

one particular department in the other data source? If certain data access rights can be derived from data sources, can it be done automatically and do specific data warehouse features create additional needs for data access control?

One of the existing approaches uses data sources users' access rights to derive the data warehouse users' access rights directly [5], [4]. In this approach the data warehouse is treated as a view over data sources, where every data warehouse user has a view over only authorized data from data sources.

Another approach [7] is based on analyzing the user's analytical business functions and the information necessary to fulfill these functions. The former data source data access rights are not taken into account, because of the new nature of the business functions. Data warehouse data can also be treated as new information, which is created in the process of integration of data source data. Some data warehouse specific functionality provided by client side tools e.g. OLAP operations drill-down, roll-up can add new restrictions to the accessibility of data, for example, the user having access to detailed data on one particular department level, can have full access to aggregated data on company level.

Another consideration, which is discussed in this paper, is at what level the data access rights ought to be implemented on, the application or the data base level?

The main criteria which solution is most appropriate in any given or particular case are further discussed in [6]. The most important issues, which need to be considered, are the number of potential users and the number of planned client side tools when developing access rights to a Data Warehouse.

In section 2 we describe some data warehouses specific characteristics and considerations about data access restrictions in data warehouses. Some definitions are given in this section and an example of data warehouse star schema is introduced.

In section 3 we present basic structure of our first solution called *"Role Based Security Model"* control model in data warehouses. Our second solution "*Dynamically Adapting Views Based Data Access Control Model"* to data access control problem is presented in section 4.

In section 5 some tests on Query response time are presented and results are briefly discussed. Finally section 6 concludes the paper and sketches the path of our future research work.

2. Data warehouse specific characteristics and restricted data access

If we compare data warehouses to on-line transaction processing systems, we can point out some distinguishing characteristics, which can result in specific requirements for data access control. The following issues for building data access control architecture in a data warehouse were taken into account [2], [3].

⊄# The data warehouse integrates data from different data sources with different data access control mechanisms, if we consider the implementation side, and with different data sensitivity level and with different scope of allowable data for each user on the information side.

⊄# The typical data access rights for users in the data warehouses are read operations.

⊄# The purpose of data usage in a data warehouse can be data analysis, knowledge discovery, and data mining. These purposes can define sets of users different from data source users. In this case the users are managers and data analysts and they are not always the users of data sources. If the reporting purpose is specified too, the users and their data access rights can be similar to data source users.

⊄# There are no limitations on the number and type of client side tools used. The client side applications can vary from reporting and OLAP tools to data mining applications. These tools don't only provide predefined reports and queries, but also the capability to query the database directly. In this case also, the users should have limited access to data. The data from different data sources are consolidated in the same physical data structures in the database. It is possible that the users of one data source operational system can have limited access to other data source data.

⊄# In the on-line transaction systems, which serve as data sources for the data warehouse, the limitations can be implemented on the application side and the server side jointly. For the data warehouse, implementation of the server side limitations leads to a more secure and consistent solution with easier administration.

A question arises however, how a consistent user access rights control system, which would derive access permissions from data sources and minimize the administration can be implemented, if data warehouse specific additional restrictions have to be defined. One possible solution is shown on Figure 1.

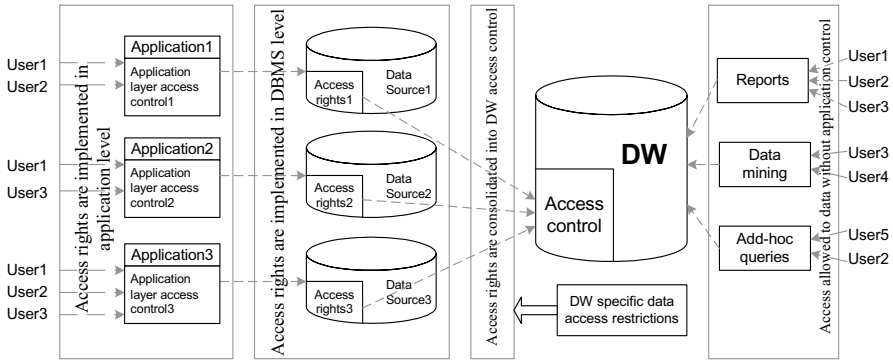


Figure 1. DW data access control architecture with derived access rights.

We have to specify some key concepts to describe two alternative models of the data access control architecture implementation.

Two possible ways of controlling data access, further referred to as horizontal restrictions and vertical restrictions:

- The **horizontal restriction** on the database table is a set of rows of the table accessible to the user.
- The **vertical restriction** is a set of columns accessible to the user.

With **data access right** we denote the ability to read data in data warehouse tables according to the horizontal and vertical restriction.

If the user has only access to a subset of all rows of the table, they should believe that the data warehouse table contains only these rows. There are some existing techniques for implementation of horizontal restrictions, e.g. views, dynamic views and virtual private database in database systems [1].

There may be a case, when a user can only access particular columns of the database table, whether the table implements a dimension or a fact table of the star schema. One possible solution for the problem is implementing the restrictions on columns in the user tools, if they have corresponding features. But this solution doesn't satisfy the direct querying needs and in case of many user tools leads to difficult or even impossible data access rights administration. Another solution is the implementation of vertical restrictions with database views or dynamic views.

We will provide a simple example of a data warehouse star schema to illustrate our solutions and show the necessity for these restrictions in Figure 2. The star schema consists of three dimension tables DEPARTMENT, TIME and PERSON and a fact table FACT_TABLE with two fact measures FACT1 and FACT2. For 'User1' the following data access rights are defined – he can see the data in all rows in all tables by departments 'Dept2', 'Dept3' and 'Dept4', he can also access all columns except the columns MARITAL_STATUS and FACT1. Data accessible to 'User1' in Figure 2 are marked dark grey. The light grey colour shows the data accessible only in one direction – vertical or horizontal.

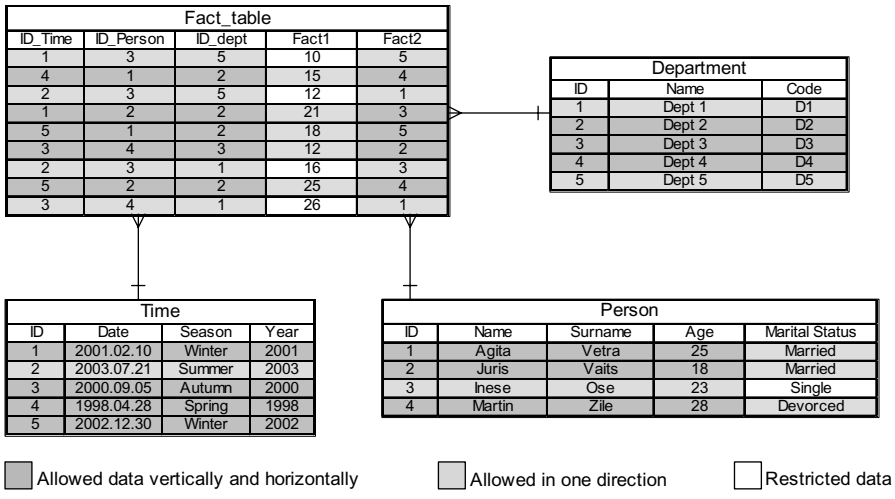


Figure 2. The example star schema.

3. Role based data access control model design

Assigning roles is one of the ways that makes implementing data warehouse data access control design possible. The role can be considered as a set of unique data access rights. These rights grant a user access to data in the data warehouse.

If we assume that the data warehouse is a set of all its data, stored in rows and columns of the database table, we can look at the data warehouse as two different data subsets – rows and columns. A role can be defined as a data set ROLE={z|x⊂H, y⊂V, z=x~ y} where H is used to denote horizontal restriction and V is used for vertical restriction. If roles are defined in this way, all users having access to the same set of rows and columns get the same role. We control user data access rights with roles, because the number of roles is usually smaller than that of data warehouse users.

We will use well-known way for implementing data access restriction. Our model is based on views, where horizontal and vertical restriction is accomplished, and database object privileges, in the way, that makes possible automatic data access control model implementation, administration and maintenance. By using further described approach, it is possible to use derived data access rights from data sources.

Each role needs its own set of views. All data access rights are built in these views. The role's physical implementation can be the traditional database views and materialized views. In the latter case, it is necessary to refresh all role based views after the data warehouse refreshment to keep data access management implementation consistent with the roles' specifications.

Not all tables, especially dimensions, need to hide data from users. For example, no restrictions exist for the time dimension. Other similar dimensions could also be found in data warehouses. Not only horizontal restrictions have exceptions. Some of the columns can be always seen for all users. These are, for example, all primary keys columns. Since primary keys for dimensions are usually automatically generated numbers and a fact table has composite primary key that consists of foreign keys to dimensions, it isn't necessary to hide them from the user. Before defining each restriction, it is important to evaluate the necessity of this restriction. Needless restrictions prolong query runtime and increase the database.

3.1. Implementation with one restricted dimension

One simple example of data access rights restriction is the case when the data warehouse star schema is restricted by only one dimension, as it is shown in Figure2.

The **restricted dimension** is a star schemas dimension table with one **restricted column** which values determine the horizontal restriction.

The accessibility of star schemas fact table depends on foreign keys values, which corresponds to restricted dimension primary keys of horizontal restriction. The accessibility of other dimension tables depends on corresponding foreign keys values of the set of allowed records in the fact table.

The dimension that horizontally restricts all data available in the star schema in Figure 2 is the DEPARTMENT dimension. The restricting column in this example is the CODE column. This means that users from one department cannot see the data

from other departments. Because companies have quite a complicated hierarchical organizational structure, horizontal restriction cannot be simple either.

Information about roles und users ought to be saved in the data warehouse. One possible solution is shown in Figure 3, which represents two tables: one for users and one for roles. The ROLES table maintains records for all roles and their data access rights on certain rows in the DEPARTMENT dimension and all accessible columns in all dimensions and fact tables.

Column ROWS in ROLES table contains the allowed values of restricted dimension's restricted column. Column COLS contains the allowed tables and columns in the star schema, and it serves as a basis for vertical restriction.

Roles

ID	Name	Rows	Cols
1	Role1	D%	Time,Department,Person,Fact_table
2	Role2	D2	Time,Department,Fact_table
3	Role3	D2,D3,D4	Time,Department,Fact_table(Fact2), Person(Name,Surname,Age)

Users

ID	Username	Role
1	User1	2
2	User2	3
3	User3	1
4	User4	3

Figure 3. Access restriction information.

The ROLES table in Figure 3 has three different roles: 'Role1' users can see all data without any exceptions. 'Role2' users can see data from 'Dept2', but dimension PERSON is hidden from them. 'Role3' is earlier demonstrated in Figure 2. Here we have to denote that we make no vertical restrictions on primary key columns.

Except for data structures, where information about users and roles is stored, it is necessary to implement the views depending on roles.

General notation for the view that corresponds to the horizontally restricted dimension is represented in the code (1). General notation for unrestricted dimensions is as follows in (2). The third and last type of data warehouse tables that needs views are fact tables. Its views generally look as in (3).

```
Create view <restricted_dimension>_<role> as select <col1>, <col2>,…,<coln>
from <restricted_dimension> where <col_restricted> like <rows_code1> or
<col_restricted> like <rows_code2> or … or <col_restricted> like <rows_coden>
                                                                        (1)
Create view <base_dimension>_<role> as select <col1>, <col2>,…,<coln> from
<base_dimension> where <primary_key> in (select <foreign_key_base_dimension>
from <facts_table>_<role>)                                              (2)
Create view <facts_table>_<role> as select <col1>, <col2>,…,<coln> from
<facts_table> where <foreign_key_restricted_dimension> in (select
<primary key> from <restricted dimension> <role>)                       (3)
```

For example, for the situation shown in Figure 2 where data access restriction for 'Role3' is given, four views have to be made in the data warehouse. Create statements according to the previously defined notation are in (4).

```
Create view department_role3 as select id, name, code from department where
code like 'd2' or code like 'd3' or code like 'd4'
Create view fact_table_role3 as select id_time, id_person, id_dept, fact2
from fact_table, where id_dept in (select id from department_role3)
Create view person_role3 as select name, surname, age from person where id in
(select id_person from fact_table_role3)
Create view time_role3 as select id, date, season, year from time where id in
(select id_time from fact_table_role3)                                    (4)
```

To automate these operations certain procedures need to be followed. After the data warehouse refreshment they can accomplish all of the above mentioned statements (1),(2),(3) and a little bit more. Let's look at the basic steps that need to be fulfilled in the following procedure.

```
If role_inserted then insert into TMP_TABLE values (role,'I','');
If role_deleted then insert into TMP_TABLE values (role,'D','');
If role_updated then insert into TMP_TABLE values (role,'U','');
If user_role_add then insert into TMP_TABLe values (User,'I',role);
If user_role_delete then insert into TMP_TABLE values (User,'D',role);
If user_role_change then begin
  Insert into TMP_TABLE values (User,'D',role_old);
  Insert into TMP_TABLE values (User,'I',role_new);
end;
For all_role_changed(TMP_TABLE) do
  If view_op in ('D','U') then drop_view(role);
  If view_op in ('I','U') then
       create_view (role,is_true_rows(role),is_true_cols(role));
End;                                                                      (5)
```

In our solution we propose to make a table TMP_TABLE (<role> or <user>, <operation>, null or <role>) for saving some temporary information. Our pseudo code (5) uses some functions, e.g. function *role_inserted* returns true if a new role is inserted into the ROLES table. Function *user_role_add* returns true if a role is added to user's record.

After the ROLES table refreshment we need to make corrections in our views based data access control system. Some other functions are needed, e.g. function *all_role_changed* gathers all records from TMP_TABLE that describe changes in the ROLES table.

Unfortunately, not all work is done by refreshing views. We have to grant SELECT rights to some users and revoke SELECT rights from other users depending on changes made in the tables ROLES and USERS. It can be accomplished by making the following basic steps:

```
For all_user_role_changed and all_role_changed in tmp_table do
  If User_op='D' then revoke_select_on (role,User);
  If User_op='I' then grant_select_on (role,User);
  If view_op='U' then grant_select_on (View_role,all_view_users(role));
End;                                                                      (6)
```

All changes in view based data access control structure can be made automatically, but the specific features of user side data access tools can sometimes add manual tasks.

3.2. Model with more than one restricted dimension

As data warehouse usually contains data from more than one data source, restrictions can be defined on more than one dimension. In previous section all horizontal restrictions were made corresponding to one dimension e.g. DEPARTMENTS dimension. We can also add horizontal restrictions, for example, on TIME dimension, which will show how old the data that the user is able to see are. For implementing the model with restriction on year, it is possible to modify the ROLES dimension a little. An example is shown in Figure 4 below:

Roles				
ID	Name	Rows_dept	Rows_time	Cols
1	Role1	D%	19%	Time, Department, Person, Fact_tabe
2	Role2	D2	2001,2003	Time, Department, Fact_tabe
3	Role3	D2,D3,D4	2%	Time, Department, Person(Name,Surname,Age), Fact_table(Fact2)

Figure 4. Roles with two restricted dimensions.

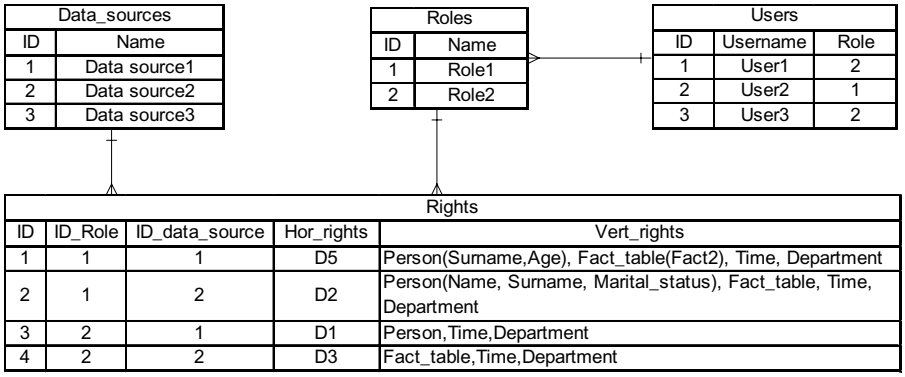
In such way it is possible to add as many dimensions as necessary. If in situation with one restricted dimension accessible rows were found starting with the restricted dimension, then in this case the restriction process starts from the fact table. So in general the fact table's view structure is:

```
Create view <facts_table>_<role> as select <col1>,…,<coln> from <facts_table>
where <foreign_key_restricted_dimension1> in (select <primary_key> from
<restricted_dimension1> where <code_rd1> like <rows_code1_rd1> or...or
<code_rd1> like <rows_coden_rd1>)
Intersection
select <col1>,…,<coln> from <facts_table> where
<foreign_key_restricted_dimension2> in (select <primary_key> from
<restricted_dimension2> where <code_rd2> like <rows_code1_rd2> or...or
code_rd2> like <rows_coden_rd2>)                                          (7)
```

All dimensions are defined in the same way as previously we defined dimensions without horizontal restriction (see section 3.1). Pseudo code (5) and (6) describing basic steps for refreshing views remain unchanged as in the previous section.

3.3. Model with data access rights derived from more than one data source

The most complicated solution is necessary when data access rights are derived from more than one data source. We propose that all these data access rights are saved in one table. Our example in Figure 5 is an illustration of this solution.



Data_sources

ID	Name
1	Data source1
2	Data source2
3	Data source3

Roles

ID	Name
1	Role1
2	Role2

Users

ID	Username	Role
1	User1	2
2	User2	1
3	User3	2

Rights

ID	ID_Role	ID_data_source	Hor_rights	Vert_rights
1	1	1	D5	Person(Surname,Age), Fact_table(Fact2), Time, Department
2	1	2	D2	Person(Name, Surname, Marital_status), Fact_table, Time, Department
3	2	1	D1	Person,Time,Department
4	2	2	D3	Fact_table,Time,Department

Figure 5. Data access restriction from more than one data source.

Horizontal restriction remains simple and is almost the same as in the model with more than one restricted dimension. Only this time, instead of intersection we have to use union. Vertical restriction is a more complex issue. Some columns need a function that returns either the cell value, if it is accessible, or an empty cell. Such function is part of dynamically changing views and will be discussed in the section 4.3.

3.4. What are the decision points?

The Role based data access control design is easy to use, if there are comparatively few roles and each role is associated with many users. If there are many roles, administration of such structure isn't an easy task. If there are changes in the role's rights, corresponding changes have to be made in view definitions. This problem could be solved automatically as described earlier in this paper in the example of the pseudo code (5) and (6), but the complexity of the process can lead to security threats. When the user's role is changed, some manual changes can be done in client side data access tools, for example changing accessible predefined reports.

We have to admit that changing role's rights isn't a good idea as it often means more users will be added. It can lead to an erroneous situation when new rights are suitable for none of the same role users, but one user can see something prohibited and data access control in the data warehouse is breached.

Role based data access control is not an appropriate solution in situations when there are many roles or almost each user has their own role. Consequently, you will build a cumbersome view structure in your data warehouse. In the case when number of users is close to the number of roles, it is better to use usernames as roles. That removes from list the tasks concerning changes in relationship user-role.

Role based data access control is a way to implement a lot of requirements for data warehouse data security by leaving query response time in previous limits, if it is possible to use materialized views for query optimization, but it can take a lot of additional administration work depending on used client side tools as mentioned in section 3.1.

4. Dynamically adapting views based data access control model

In the previous section the implementation of the described method has some administration difficulties in the case of many warehouse users and roles. We have developed another method, where there is no need for administration and huge view structures. Each base table has one view and this view contains available data depending on each user's rights.

By **dynamically adapting views** we are denoting views fulfilled with only those data, which are accessible in the base table to a corresponding user.

If the user has no access to any data in the base table, the defined corresponding view is blank. The view can be blank in two situations:

- The user has no access to any rows in the base table;
- The user has no access to any columns in the base table.

For security reasons the blank views have to be hidden from the user. It can be easily made on the database level by removing SELECT rights from a particular user. More problems can arise connected with client side data access tools, when it is possible that blank objects are visible. Nevertheless, there are tools where this problem is solved.

By defining data access control plan for dynamically adapting views, we always have to keep in mind that every restriction affects query performance.

4.1. Dynamically adapting views with one restricted dimension

To describe this implementation model, we would like to return to the data warehouse example schema in Figure 2 and to added security tables in Figure 3. Only this time, we remove column ROWS from the ROLE table and add it to USERS table. This change has the following goals:

- To separate the horizontal restriction from the vertical restriction. In this approach roles are used only for defining the vertical restriction. We tried to avoid using the vertical restriction where possible, as it impacts on the query performance and makes indexes and materialized views impossible to use.
- In this way the number of roles is much smaller so it is possible to make a materialized view for each role, where the vertical restriction is necessary. It decreases query execution time as no vertical calculation is needed during query execution time.

In the implementation of dynamically adapting views we followed three steps.

1.step. One of them is creating a function for the horizontal restriction. This function is included in the horizontally restricted dimension view definition in such way that this views returns only user's accessible records. If there is no additional vertical restriction on the horizontally restricted dimension the general form of the corresponding view is (8). In our example in Figure 2 such view definition is only for the DEPARTMENT dimension. It looks as follows in (9).

```
Create view <restricted_dimension>_view as select * from
<restricted_dimension> where F1(<restricted_col_val>,<user>,
<rows_col_name>)=1                                                    (8)
Create view DEPARTMENT_VIEW as select * from DEPARTMENT where
F1(CODE,user,ROWS)=1;                                                 (9)
```

The main structure of Function F1 is quite simple. The Function has three input parameters, a restricted column value, username and the name of the column in USERS table where accessible rows are stored.

```
Select rows_col_name from USERS where username=user;
If restricted_col_val=substr(rows_col_name) then return 1 else return 0;  (10)
```

2.step. The next step is creating the vertical restriction. This part is more complicated and consists of many similar functions. The purpose of these functions is either to return columns cell value or a blank cell, if the column isn't accessible. These functions can be divided into two groups:

a) *Universal functions* that are used in the dimension views' definitions. As an input parameter they get a username, a column name, a dimension name and records primary key, which in our solution is an automatically generated number in the ID column. Because data types for returned value could be NUMBER, DATE and CHAR, it is impossible to implement them with only one function.

b) *Fact table functions*. Each fact table needs its own functions. Their number depends on data types used in the fact table. Almost all fact tables have different primary keys that depend on the related dimension number. For this reason fact table functions as input parameters don't get a table name, but receive the whole composite primary key instead.

The main structure of the vertical restriction on columns can be made in three steps.

```
1.  Select col into cell_value from base_table where
    primary_key=primary_key_columns;
2.  Select COLS into access_cols from ROLES, USERS where username=user and
    roles.id=users.role;
3.  If col=substr(access_cols) then return cell_value else return null;  (11)
```

If functions for the horizontal and vertical restriction are created, general structure of the fact table's view is as in (12). Dimensions' views are defined as follows in (13).

```
Create view <fatct_table>_view as select <primary_key_columns>,
<func(<user>,<col1>,<primary_key_columns>)>,...,<func(<user>,<coln>,
<primary_key_columns>)> from <fact_table> where
<foreign_key_restricted_dimension> in (select <primary_key> from
<restricted_dimension>_view)                                          (12)
      Create view <base_dimension>_view as select <primary_key>,
<func(<Username>,<PK>,<col1>,<base_dimension>)>,...,<func(<Username>,<PK>,
<coln>,<base_dimension>)> from <base_dimension> where <primary_key> in (selec
<foreign_key_base_dimension> from <facts_table>_view)                 (13)
```

In our example from Figure 2 the views for fact tables and persons are as follows, by assuming that the DEPARTMENT dimension has no vertical restriction but the time dimension is open for all:

```
Create view FACT_TABLE_VIEW as select ID_TIME, ID_PERSON, ID_DEPT,
col_fact_numb(user,'fact1',ID_TIME,ID_PERSON,ID_DEPT) as FACT1,
col_fact_numb(user,'fact2',ID_TIME,ID_PERSON,ID_DEPT) as FACT2, from
FACT_TABLE where ID_DEPT in (select ID from DEPARTMENT_VIEW);
Create view PERSON_VIEW as SELECTID, col_dim_varch(user,ID,'name','person') as
NAME, col_dim_varch(user, ID,'surname','person') as SURNAME,
col_dim_varch(user,ID,'age','person') as AGE,
col_dim_varch(user,ID,'marital_status','person') as MARITAL_STATUS from PERSON
where ID in (select ID PERSON from FACT TABLE VIEW).                    (14)
```

3.step. When dynamically adapting views are made, a procedure for grants is also necessary. This procedure controls the existence of blank views for any user. If there are any, the procedure revokes SELECT rights from the corresponding user on the corresponding view. The general steps of this procedure are:

```
For all_users except owner do revoke select on tablename_view from user;
For all_users except owner do
  If has_data(tablename_view,user) then
        grant select on tablename view to user                          (15)
```

If the data access control model is implemented in this way, there is no need for additional administration. We have to add new users and roles but all further steps are done automatically.

4.2. Dynamically adapting views with more than one restricted dimension

The idea behind this solution has already been described by and large in section 3.2. However, some changes will have to be added. Columns with restriction on rows are moved from the ROLES table to the USERS table. The rationale for doing this was described in the previous section. The system of defining views also remains the same. Some changes are made in the fact table view general definition.

```
Create view <facts_table>_view as select <primary_key_columns>,
<func(<username>,<col1>,<primary_key_columns>)>,...,<func(<username>, <coln>,
<primary_key_columns>)>from <facts_table> where
<foreign_key_restricted_dimension1> in (select <primary_key> from
<restricted_dimension1> where
F1(<restricted_col_val>,<user>,<rows_col_name1>)=1)
Intersection
Select <primary_key_columns>,
<func(<username>,<col1>,<primary_key_columns>)>,...,<func(<username>,
<coln>,<primary_key_columns>)>from <facts_table> where
<foreign_key_restricted_dimension2> in (select <primary_key> from
<restricted_dimension2> where
F1(<restricted_col_val>,<user>,<rows_col_name2>)=1)                     (16)
```

All dimensions are defined in the same way as horizontal restriction free dimensions were defined previously (see section 4.1).

4.3. Dynamically adapting views with more than one data source

The system of saving rights form multiple data sources is perfectly illustrated by the model in Figure 5. To allow 'User2' view the information shown in Figure 6 we made some changes in our functions that are responsible for the vertical restriction. First, we define the fact table with a structure as in the previous section. Only this time instead of intersection we use union. Dimensions are also defined as in the previous section. Functions responsible for the vertical restriction return cell value, if this cell is contained in any pair <hor_rights, vert_rights> for this role, if not these functions return blank cell.

Basic steps for the function implementing the vertical restriction in this case are as follows:

```
1.Select col into cell_value from base_table where
primary_key=primary_key_columns;
2.Corsor C1 is select * from rights, roles, users where
rights.id_role=roles.id and roles.id=users.role and username=user
3.For each_cursor_record do
If contains(col,primary_key_columns,hor_rights,vert_rights) then return
cell_value else return null;                                          (17)
```

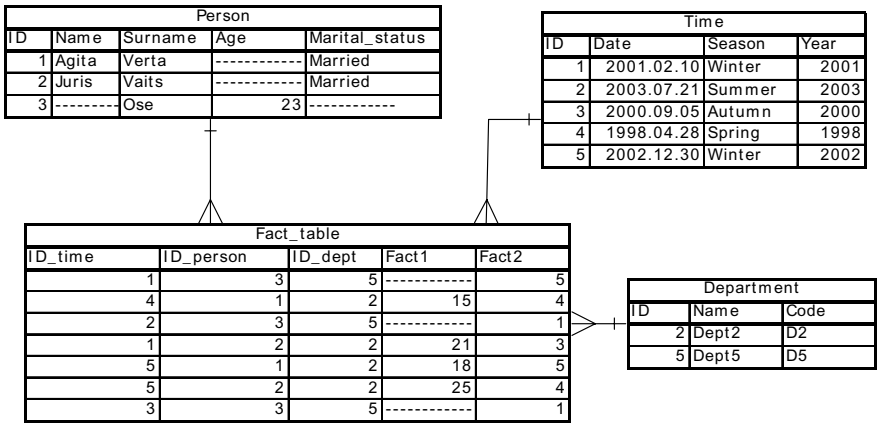


Figure 6. Access restriction from multiple data sources.

5. Models performance testing

After development of our proposed models, we made some tests on query response time. We run them on our data warehouse data mart that consists of eight dimensions. The largest of them are persons with 75000 records, orders with 60000, education with 11000 and addresses with 17000 records. Fact table consists of 450 thousands of rows. We took two queries from typically used user reports. We didn't use any aggregate tables, materialized views or indexes for improving query response time. Queries were run on views, described in previous sections. We run these queries from two different users DISC1 and DISC2 simultaneously. These

users were made as users from different departments with different data access rights.

The results are included in Figure 7.

	Disc1 role based DAC	Disc2 role based DAC	Disc1 adapting views	Disc2 adapting views	Full base tables
Query1	0.93	0.78	76.46	64.34	1.20
Query2	0.13	0.18	6.43	5.67	0.18

Figure 7. Models performance testing.

Tests were completed on Oracle 9.0 by using Oracle Trace utility. Each query is presented with CPU time used. Query1 consisted from three tables, aggregation on one column and the results were grouped by one field. Query2 was more complicated. It used data from five tables, had additional conditions, sorting, grouping, and aggregation on three columns and contained union operation.

As it is seen from Figure 7 and is mentioned in this paper, role based data access control mechanism uses less CPU time than dynamically adapting views. Future query testing using appropriate indexes and materialized views is concerned.

6. Conclusions

In this paper we introduced two solutions for implementing data warehouse security policies. The Role based security model is easy to understand and can be made in such a way that query response time isn't seriously impacted. From the other side this technique can lead to a huge database and a complicated views structure that needs a lot of administration work. The other solution is dynamically adapting views that have opposite characteristic features from the Role based security model. This solution needs almost no administration, but as the views are made dynamically, query response time can increase dramatically, if no additional work is done.

We suggest using the role based security system, if the data warehouse doesn't have a lot of roles and there are resources for administrating all objects that need to be created by using this model. Dynamically adapting views are more appropriate if the data warehouse has many users with different access rights and there are no resources for security administration.

A lot of restriction methods are already built in DBMS and data access tools. For this reason it is possible to use the implemented methods. It is a good decision, if the security policy requirements are not complicated, for example, only horizontal restriction on one or more dimensions is required. But if in the security policy vertical restriction is necessary, there will be problems of finding use for the built-in security mechanism.

Our future work will have to deal with making improvements to our methods. The basic guideline is making queries to return data faster. This can be done in several ways:

- by adding rights indicator to each record in each table;

- by using built-in access restriction functions like virtual private database for separating the horizontal restriction from the vertical one;
- by using materialized views.

References

[1] K.B.Edwards, and G.Lumpkin. Security and the Data warehouse. Oracle Corporation, http://otn.oracle.com/ow2003/ow_security.html , 2003.
[2] W.H.Inmon. Building the Data Warehouse, John Wiley, 1996.
[3] R.Kimball, and M.Ross. The Data Warehouse Toolkit, John Wiley & Sons, 2nd edition, 2002.
[4] A. Rosenthal, and E. Sciore. View Security as the Basis for Data Warehouse Security. Proceedings of the Intemational Workshop on Design and Management of Data Warehouse DMDW'2000, Sweden, June 2000.
[5] A. Rosenthal, and E. Sciore. How Can Data Sources Specify Their Security Needs to a Data Warehouse? IEEE Workshop on Security in Distributed Data Warehousing, October 2001.
[6] C. Silbernagel. Data security: Protecting the warehouse from within. DM Direct, June 1999.
[7] E. Weippl, O. Mangisengi, W. Essmayr, F. Lichtenberger, and W. Winiwarter. An Authorization Model for Data Warehouses and OLAP. Proc. of the Workshop on Security in Distributed Data Warehousing. New Orleans, Louisiana, October 2001.

SPECIFICATIONS

SCIENTIFIC PAPERS UNIVERSITY OF LATVIA. 2004. VOLUME 672, Datbases and Information Systems
309–320 P.

Use of Knowledge Engineering Techniques for Creation and Analysis of Aggregate Specifications

Henrikas Pranevicius, Germanas Budnikas

Business Informatics Department, Kaunas University of Technology
StudentÔ56-301, LT-3031 Kaunas, Lithuania
e-mail: hepran@if.ktu.lt

Abstract. The paper presents a technique that applies knowledge engineering techniques for creation of Aggregate specifications. Application knowledge base (KB) is created using the knowledge acquisition technique joined with a piece-linear aggregate model. The production rules of the application KB are transformed to single hit decision tables, and the static properties of the KB are checked in Prologa system. Further, the application KB is combined with the defined KB of validated properties and validation method, and application KB dynamic properties are checked in the expert system in CLIPS. A validated application KB is used defining a framework of Aggregate specification using Praxis editor and supplementing Praxis generated framework with the application functional description. The technique is illustrated with an example of a single channel queuing system.

Keywords. Aggregate specification, knowledge base, single hit decision table, expert system, and validation of static and dynamic properties, CLIPS, PROLOGA.

1. Introduction

Formal methods and specification languages are widely used for design of distributed systems. The most popular formal specification languages being used for description of distributed systems are SDL, Lotos, Estelle.

Specification language Estelle/Ag is based on Aggregate method [10]. There are some differences between Estelle/Ag and Estelle—the piece-linear aggregate model is used in Estelle/Ag. The use of such a model instead of a finite-state automaton, which is the formal background of standard Estelle, enables to create validation and simulation models based on a single specification. This is possible due to the special structure of the piece-linear aggregate. Further in the paper we use notion of Aggregate specification as the one written in Estelle/Ag language.

A construction of specifications in this language is performed in two phases. Specification editor Praxis helps to define the specification framework that describes an interaction of specified system aggregates, their states and conditions of state change. Next, the framework is heuristically supplemented with the knowledge from a conceptual model about behaviour of an analysed system.

There are a lot of works where knowledge-based systems are used for creation of (formal) specifications. A key feature of techniques used in such a works is the

analysis of correctness that is performed both during the construction of the initial knowledge description and while a target model or KB has been created.

This paper presents an approach that utilises techniques of knowledge representation as well as knowledge engineering for building up a knowledge base (KB) aimed at transformation to Aggregate specification and validation* of the KB static and dynamic properties. A transformation phase to the specification is described too.

The static properties are characteristics of a KBS that can be evaluated without its execution. Such an evaluation is often referred to as static verification. During static verification, a KB is checked for anomalies. Preece and Shinghal [12] present a classification of the anomalies that may be present in rule-based systems. It is necessary to note the difference between an anomaly and error. The anomaly indicates the existence of a possible error. The dynamic properties are those characteristics of a rule-based system that can be evaluated only by examining how the system operates at a run time.

Knowledge base and Aggregate specification are used for description of the same problem. Many authors (e.g. Bruynooghe *et al*. [2]) believe that the declarative style of description that is used in KBs is more understandable and acceptable than the procedural style (the latter is used in Aggregate specifications). This is because a problem is described at the knowledge level at which the knowledge engineer specifies expertise during knowledge acquisition. Therefore, the knowledge description is presented not using strict mathematic notation but concepts of an application domain that is natural. Due to this reason, we suppose that the creation of Aggregate specifications using KB is more attractive in that sense.

Until now, knowledge-based techniques were not used for the creation of Aggregate specifications except Praxis system. However, this system generates structure of the specification only and does not define the behaviour of an application. Our approach does it by using KB.

Our technique is similar to the one proposed by Fuchs and Schwitter [4] in such a way that they also offer transformation of problem domain description to representation structures and then to an executable language. However, our technique checks general properties during validation and verification while Fuchs and Schwitter [4] technique checks specific invariant properties. Our approach is similar to that of [15] since they both offer the use of declarative languages for a description of the user needs. We use aggregate model concepts for representation of the object structure, while Specht [15] use “object as theory” model. In our approach, in contrast to the compared one, we emphasise validation of the created declarative description. A transformation to target representation is used in both approaches. Our approach is similar to Arentze *et al.* [1] by the use of decision tables (DT) for representation of state-based systems. Our approach as well as many others, example of which is [3], exploit advantages of tabular representation in order to perform verification by transforming certain representation to DTs. However, our approach is specific in the sense that verification is performed on knowledge base that is oriented at creation of Aggregate specifications. In [9], already created Aggregate specifications are validated using first order predicate logic and Prolog. While in our work we use knowledge techniques for analysis of the specifications to be created using knowledge

* In the paper, for the sake of briefness, sometimes we refer to validation having in mind both validation and verification because "validation subsumes verification" [11].

base. Sellini *et al*. [14] and we use intra- and inter- validation for analysis of the acquired knowledge. They also use an intermediate formalised description (application KB in our case) for construction of a knowledge model (specification in our case). While performing static verification of an application KB, we use results of Vanthienen *et al*. [18], whereas when analysing the dynamic properties we use the reachable state method that is similar to the functional validation method suggested by Preece *et al*. [11] in a view of analysis of execution paths.

An applicability of our technique is defined by the applicability of the PLA method and Aggregate specifications. The technique is oriented for creation of Aggregate specifications which primary domain of usage is communication protocols and business systems. Our technique was successfully applied for the creation of Aggregate specifications of the single channel queuing system, alternating bit protocol and a network of queuing systems. The scalability of the proposed technique is limited by software tools that are used in creating the specification.

The paper is structured as follows. The main stages of the proposed scheme of Aggregate specification creation are presented in section 2. Section 3 describes construction of the application KB and analysis of its static and dynamic properties. Section 4 describes procedures for generation of specification structure with Praxis and addition of functional description of the application from the KB to the generated structure. Section 5 presents an example of the proposed technique. Conclusions sum up the proposed approach.

2. The main stages of the proposed scheme

The developed approach is depicted in Fig.1. The application KB is created using the knowledge acquisition technique that was adapted for the creation of the specific KB, which is referred to as KB_{Ag} too and intended for mapping to Aggregate specification. Knowledge about the problem domain is represented in the KB_{Ag} in the context of the PLA model.

Production rules of the KB_{Ag} are transformed to single hit decision tables in Prologa, and static verification is performed in this system. The Prologa system is an interactive design tool for computer-supported construction and manipulation of decision tables. The system offers design techniques and additional features to enhance the construction and validation of decision tables [16]. The verification in Prologa is implemented using the tabular verification method [18] that belongs to a group of static verification methods. The transformation to Prologa DTs is specific with respect to PLA model and employs some of its concepts—aggregates, internal and external events, input and output signals, discrete state component coordinate, etc. Production rules are transformed to the DTs of certain groups thus enabling to fully exploit advantages of tabular representation to perform static verification. Functional validation is performed using the expert system (ES) in CLIPS. CLIPS (C Language Integrated Production System) is a tool for productive development and delivery of expert systems [5]. The ES is constructed by combining the KB_{Ag} with the KB of validated properties and validation method (KB VPVM). The functional validation is implemented using the reachable state validation method that can be classified as being a member of the group of formal proof methods. Further,

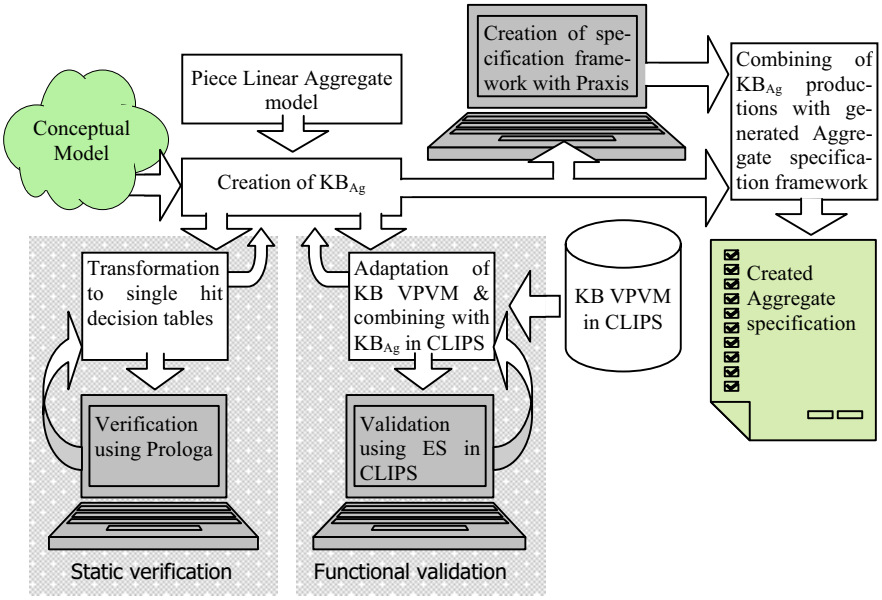


Figure 1. Scheme of the proposed approach for creation of Aggregate specifications.

using validated and verified KB_{Ag} one defines Aggregate specification framework during the session with specification editor Praxis. The generated framework is supplemented with knowledge about system behaviour extracted from the KB_{Ag} using defined mappings.

A distinctive feature of our approach is the fact that validation and verification task is performed at the initial stage of Aggregate specification creation. Validated knowledge is used both for creation of the specification framework and for supplementation to the framework. In addition, our approach defines the creation of Aggregate specifications.

3. Creation and analysis of KB aimed at transformation to Aggregate specifications

KB_{Ag} represents problem domain knowledge using production rules. This representation is chosen due to its similarity to Aggregate specification language constructions describing conditions for state change: `when` *condition* `begin` ... `end`. Since most of the common verification and validation problems in rule-based systems can be solved using DTs [16] and the tabular verification method is computerised in Prologa system, in our approach static verification will be performed using this method. Thus, in order to perform the static verification of the KB_{Ag}, its production rules have to be transformed to Prologa DTs. Moreover, as stated in [16], a DT is equivalent to a set of production rules, and their transformation to the tables can be performed without too much effort.

Because the KB_{Ag} will be used for the creation of Aggregate specifications, it has to contain knowledge about the PLA model. To acquire this knowledge, we applied

the knowledge acquisition technique [13] that was adapted for our needs. The following knowledge about the PLA model is used.

- Succeeding concepts: (a) Aggregates; (b) Input and output signals and their components, coordinates of discrete and continuous state components.
- Relations: (a) Interconnection scheme of aggregates; (b) Changes of state and signal outputs.

In our approach, the predicates and production rules are used for description of the above-mentioned concepts and relations. Applying our approach for specification of an application one has to use the defined predicates and production rules for problem representation. Thus, the KB_{Ag} is created by filling in the knowledge about the application as predicates and production rules of the defined form. Next, we present examples of predicates and productions of KB_{Ag}.

(ArrivalOfInputSignal $an_i\ iip_i^j\ x_i^j\ x_j^1\ 2\ x_j^{c_{ic}^i}$),

where an_i – symbolic name of the ith aggregate; iip_i^j – interaction point, $x_j^1, 2, x_j^{c_{ic}^j}$ – components of signal x_i^j.

(State $an_i\ w_i^1\ 2\ w_i^{c_{cc}^i}\ d_i^1\ 2\ d_i^{c_{dc}^i}$),

where $w_i^1,...,w_i^{c_{cc}^i}$ and $d_i^1,...,d_i^{c_{dc}^i}$ are the coordinates of continuous and discrete state components.

The production rule that describes state change and signal output after occurrence of an external event has the following general form:

IF (ArrivalOfInputSignal $an_i\ iip_i^j\ x_i^j\ x_j^1\ 2\ x_j^{c_{ic}^i}$)
AND (State $an_i\ w_i^1\ 2\ w_i^{c_{cc}^i}\ d_i^1\ 2\ d_i^{c_{dc}^i}$)
AND (Aux $an_i\ w_i^1\ 2\ w_i^{c_{cc}^i}\ d_i^1\ 2\ d_i^{c_{dc}^i}$)
THEN (State $an_i\ w_i^1{*}\ 2\ w_i^{c_{cc}^i}{*}\ d_i^1{*}\ 2\ d_i^{c_{dc}^i}{*}$)
AND (OutputSignal $an_i\ oip_i^j\ x_i^j\ x_j^1\ 2\ x_j^{c_{oc}^j}$))

where (Aux $an_i\ w_i^1\ 2\ w_i^{c_{cc}^i}\ d_i^1\ 2\ d_i^{c_{dc}^i}$) describes auxiliary conditions $LC_1, 2, LC_{p_m}$ that check values of coordinates of state components:

(Aux $an_i\ w_i^1\ 2\ w_i^{c_{cc}^i}\ d_i^1\ 2\ d_i^{c_{dc}^i}$) $\sum LC_1$ AND|OR…AND|OR LC_{p_m}, where $i \mid \overline{1,n}, p_m$ – the number of additional logical conditions for the m-th production rule that describes state alteration.

3.1. Static verification of the KB_{Ag}

Most of the validation problems in rule-based systems like redundant, ambivalent, categorised, cyclic or missing rules, redundant conditions, unused action parts may be resolved using DTs [16].

Our technique uses DT representation for static verification of KB_{Ag} since DTs clearly demonstrate incompleteness and in-consistency of knowledge [3] and soft-

ware tool Prologa that assists verification process. Anomalies in DTs have direct correspondence to anomalies in production rules, which make the KB_{Ag}.

Static anomalies in rule bases and decision tables

The decision table DT is defined [18]:

$$DT: CS_1 \Delta CS_2 \Delta \ldots \Delta CS_m \Downarrow AV_1 \Delta AV_2 \Delta \ldots \Delta AV_n ,$$

where CS_i is a set of condition states, AV_j is a set of action values.

The most important criterion when distinguishing tables is the question whether all columns are mutually exclusive (*single hit* versus *multiply hit*). In a single hit table, in contrast to multiply hit table, each possible combination of condition can be found in exactly one and only one column.

The following anomalies are distinguished: *intra*-tabular, which occur in a single DT, and *inter*-tabular anomalies, that originate from interactions between several DTs [6]. A relation between *intra*-tabular anomalies and anomalies in rule bases is depicted in Fig.2. A classification of inter-tabular anomalies can be found in [17].

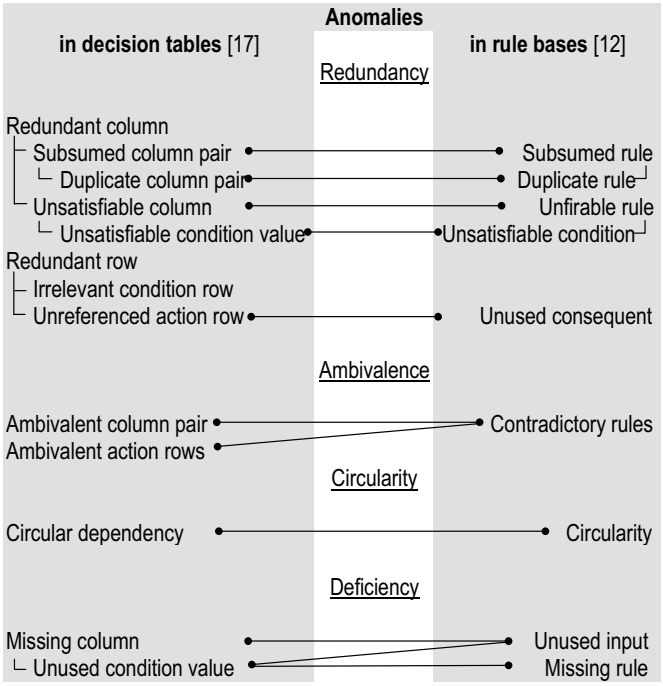


Figure 2. Relation between intra-tabular anomalies in DTs and anomalies in rule bases.

Transformation of KB_{Ag} productions to single hit decision tables

In our technique, static verification of KB_{Ag} is performed in Prologa system. The system uses single hit DT representation. Next, we present an outline of suggested steps of transformation of KB_{Ag} productions to single hit DTs.

1. Productions that describe certain types of events are transformed to corresponding event tables;

2. Predicates of antecedent (consequent) part of a rule are written in a form of condition (action) subjects in a DT;
3. Decrease of a coordinate of discrete state component by a constant value is marked with not (d_i^j | d_i^j 2 1). This notation is used while ambivalence property is being checked.

Technique for detection of intra-, inter- tabular anomalies in KB_{Ag}

This technique is mostly based on works of [16, 18] and operating in Prologa system. Below we present a portion of this technique.

A *subsumed column pair* (a case of intra-tabular **redundancy**), which definition is as presented below [6]:

> A DT contains a *subsumed column pair*, if and only if it includes a pair of columns (CS_j; AS_j), (CS_k; AS_k) (1 Ωj, k Ωt), j ∏k for which: $CS_j \geq CS_k$ and $AS_j \neq AS_k$,

in a single hit DT is represented by single column that corresponds to several KB_{Ag} productions. Thus, the anomaly of the subsumed column pair is detected if several KB_{Ag} productions correspond to the same DT column.

Full description of the technique for anomaly checking in KB_{Ag} as well as transformation steps of KB_{Ag} productions to single hit DTs can be found in [8].

3.2. Functional validation of the KB_{Ag}

Functional validation of the application KB is carried out using the expert system in CLIPS. It is built by combining already verified application KB with a KB of validated properties and validation method.

KB of validated properties and validation method and its join with KB_{Ag}

In our approach, we define the KB of validated properties and validation method that is implemented in CLIPS. It may be used for various kinds of applications with minor adaptations. The instances of dynamic properties whose validation is implemented in the KB VPVM are the absence of static deadlocks, final state reachability, boundedness, and completeness. The reachable state validation method is realised in the KB VPVM. In a view of analysis of the execution paths, this method is similar to functional validation method suggested by Preece *et al*. [11] that analyses the sequences of rules that must fire to achieve a goal.

When performing the join of KB_{Ag} with KB VPVM, the needed adaptation changes are minor—they are an adaptation of description of validated properties for a specific application. For instance, in order to check the absence of the static deadlock property, the adaptation includes definition of specific continuous state coordinates; in order to check boundedness property, individual bounds on discrete state component coordinates have to be defined. Having combined the KB VPVM with the KB_{Ag}, the ES in CLIPS is built.

Validation of the KB_{Ag} using the expert system in CLIPS

The expert system in CLIPS consists of the combined KB VPVM and the KB_{Ag}, and CLIPS inference engine that is based on the forward chaining strategy [7]. In order to perform the functional validation using the ES, the initial and the final states

of an analysed application are defined. An application model operates according to the reachable states method. If the validated properties are violated, the expert system generates a corresponding report and a designer corrects the KB accordingly.

4. Forming the specification

In the first step, specification framework is constructed during a session with the specification editor Praxis [10] using the knowledge extracted from the validated and verified application KB.

In the second step, the generated specification framework is extended by adding with the behavioural description taken from the application KB. The supplementation process consists of the following phases:

- Finding the rules in the KB that correspond to Aggregate specification constructions. These constructions describe either initial state of an aggregate, or condition of state change due to an event. Mapping between the rules in the KB_{Ag} and Aggregate specification constructions is presented in Table 1;
- Transformation of antecedents and consequents of the found rules to Aggregate specification operators. The corresponding mapping has been defined too. A part of it is given in Table 2;
- Supplementation of specification constructions with the operators resulted from the previous phase.

Table 1. Mapping between productions in KB_{Ag} and Aggregate specification constructions.

Predicates in antecedent part of KB_{Ag} productions	*Specification constructions*
(Initialize)	`Initialize` `  begin … end;`
(EndOfOperation an_i *operation*)	`when eop.`*operation* `  begin … end;`
(ArrivalOfInputSignal $an_i\ iip_i^j\ x_i^j\ x_j^1\ 2\ x_j^{c_{ic}^i}$)	`when` $iip_i^j.x_i^j$ `  begin … end;`

Table 2. Fragment of mapping between predicates in consequent part of productions in KB_{Ag} and Aggregate specification operators.

Predicates in production rule consequent	*Specification operators*
(OutputSignal $an_i\ oip_i^j\ x_i^j\ x_j^1\ 2\ x_j^{c_{oc}^j}$)	`Output` $oip_i^j\ (x_i^j, x_j^1, 2, x_j^{c_{oc}^j})$;
w_i^j = on	`Start` w_i^j;
w_i^j = off	`Cancel` w_i^j;

5. Example

We illustrate creation of inconsistent and redundant fragment of a KB_{Ag} of the single channel queuing system (QS) with priorities. Conceptual description for the described fragment states that two types of requests may arrive to a queuing system: with high or low priority. The system has queue counters for the corresponding requests. The arrived request to the empty system is started to serve. If service operation is active in the system, the arrived request is placed to the corresponding queue. Inconsistencies and redundancies are detected during the static verification. A portion of a formed Aggregate specification from queuing system KB_{Ag} is presented too.

The fragment of the KB_{Ag} of QS

r1 "system is idle, low piority request has arrived"

IF (ArrivalOfInputSignal ip2 ReqLP)
AND (State *service serv_req_pr hpqc lpqc*)
AND *service* = off
AND *hpqc* = 0
AND *lpqc* = 0
THEN *service* = on
AND *serv_req_pr* = 0
AND (State *service serv_req_pr hpqc lpqc*)

where *service* stands for service operation, *serv_req_pr* - a priority of served request, *hpqc* and *lpqc* - high and low priority queue counters, respectively.

r2 "request is being served, low priority request has arrived"

IF (ArrivalOfInputSignal ip2 ReqLP)
AND (State *service serv_req_pr hpqc lpqc*)
AND *service* = on
THEN *lpqc* =*lpqc* + 1
AND (State *service serv_req_pr hpqc lpqc*)

r2a "low priority request is being served, and low priority request has arrived"

The rule is identical to r2, except an antecedent part that is supplemented with the condition *serv_req_pr* = 0.

r2b "high priority request is being served, and high priority request has arrived"

The rule is identical to r2, except an antecedent part that is supplemented with *serv_req_pr* = 1 condition.

r2c "high priority request is being served, and low priority request has arrived"

The rule is identical to r2b, except a consequent part where $lpqc = lpqc + 1$ is changed with $lpqc = lpqc - 1$.

Static verification of KB_{Ag} of QS

While illustrating a detection of intra-tabular anomalies, only a group of some productions were transformed to a single table. This was done in order to show a fewer number of anomalies in a DT at the same time.

Redundancy - *subsumed column pair.* Productions r1, r2, r2a and r2b are used for illustration of this kind of anomaly. r12ab single hit DT is presented in Fig.3.

r12ab						
1. ^(State *service serv_req_pr hpqc lpqc*)	True					False
2. ^(ArrivalOfInputSignal ip2 ReqLP)	True				False	-
3. *service*	on	off			-	-
4. *hpqc*	-	=0		>0	-	-
5. *lpqc*	-	=0	>0	-	-	-
6. *serv_req_pr*	-	-	-	-	-	-
1. *service* = on	.	x	.	.	.	.
2. *serv_req_pr* = 0	.	x	.	.	.	.
3. *lpqc* = *lpqc* + 1	x	.	.	.	.	.
4. (State *service serv_req_pr hpqc lpqc*)	x	x	.	.	.	.
	1	2	3	4	5	6

Figure 3. r12ab single hit DT.

In this table, production r1 is described by the 2nd column, productions r2, r2a and r2b are described by the 1st column. Regardless of the possible states (0 or 1) of the 6th condition subject, a set of corresponding actions is the same. Due to this reason the condition subject state is marked with "–" (irrelevant) symbol.

Productions r2, r2a and r2b are represented by the same 1st column and the production r2 has the least number of condition subjects. This means, the production r2 subsumes r2a and r2b, and thus these two rules are redundant.

Irrelevant condition row anomaly is illustrated by the r12ab DT that is presented in Fig.3. The 6th condition subject of this DT is irrelevant and redundant. It means the condition of r2a and r2b productions that checks *serv_req_pr* value is redundant.

r12bc							
1. ^(State *service serv_req_pr hpqc lpqc*)	True						False
2. ^(ArrivalOfInputSignal ip2 ReqLP)	True					False	-
3. *service*	on		off			-	-
4. *hpqc*	-		=0		>0	-	-
5. *lpqc*	-		=0	>0	-	-	-
6. *serv_req_pr*	0	1	-	-	-	-	-
1. *service* = on	.	.	x	.	.	.	.
2. *serv_req_pr* = 0	.	.	x	.	.	.	.
3. *lpqc* = *lpqc* + 1	.	?	.	.	.	.	.
4. (State *service serv_req_pr hpqc lpqc*)	.	x	x	.	.	.	.
	1	2	3	4	5	6	7

Figure 4. r12bc single hit DT.

Ambivalence - *ambivalent column pair* anomaly. Productions r1, r2b and r2c are used for anomaly illustration. r12bc single hit DT is presented in Fig.4. The 2nd column of this table corresponds to productions r2b and r2c. A cross of the 3rd row of an action subject and the 2nd column of the table is marked by "?" symbol. It means that these two productions define ambivalent actions, namely *lpqc* = *lpqc* + 1 and not (*lpqc* = *lpqc* + 1), while their conditions are overlapping. This anomaly indicates an

occurrence of a typing error when defining the action—instead of "+" sign "–" sign was specified. After fixing this error, the duplicate column pair anomaly is indicated—productions r2b and r2c, which correspond to the 2nd column of DT in Fig.4, are identical.

A portion of a formed Aggregate specification

A fragment of the created Aggregate specification of the QS example using our technique is presented below. The description of the QS behaviour in case of arrival of a low priority request is included in the productions r1 and r2. These rules have been used to supplement `when ip2.ReqLP` construction of Aggregate specification framework. The Praxis generated code is written in `typewriter` font, the added code—in *italic*.

```
Trans
 when ip2.ReqLP
  begin
    if ((service = off) and (hpqc = 0) and (lpqc = 0)) then
       begin
         serv_req_prior = 0;
         Start service;
       end;
    if (service = on) then
       begin
         lpqc = lpqc + 1;
       end;
  end;
```

6. Conclusions

Summarising the paper we would like to emphasise the following:

- This paper has showed a possibility to perform static verification of Aggregate specification that is expressed in a knowledge base;
- The use of KB defines the creation of Aggregate specifications. The production rules representation technique combined with PLA model is used for description of application structure and behaviour. The description at the knowledge level permits to operate not using strict mathematic notation but the concepts of the application domain;
- KB validation and verification permits to perform analysis of the specification under creation at its initial construction stage.

7. References

[1] Arentze, T.A., A.W.J. Borgers, H.J.F. Timmermans. The integration of expert knowledge in decision support systems for facility location planning, *Computers Environment and Urban Systems*, 19(4), 1995: 227-247.

[2] Bruynooghe, M., N. Pelov, M. Denecker. Towards a more declarative language for solving finite domain problems. In *Proc. of the ERCIM/ COMPULOG Workshop on Constraints*. 1999: 1-14.

[3] Degelder J; M. Steenhuis. A knowledge-based system approach for code-checking of steel structures according to Eurocode 3. *Computers-and-Structures*. 67(5), 1998: 347-355.
[4] Fuchs, N.E., R. Schwitter. Attempto Controlled English. In *Proc. of the CLAW 96, The First International Workshop on Controlled Language Applications*, Katholieke Universitet Leuven, 1996.
[5] Giarratano, J.C. *CLIPS Reference Manual. Basic Programming Guide, CLIPS Version 6.0*. Software Technology Branch, Lyndon B. Johnson Space Center, JSC-25012, 1993: 388 pp.
[6] Mues C. *On the Use of Decision Tables and Diagrams in Knowledge Modeling and Verification*, PhD dissertation, Katholieke Universitet Leuven, 2002: 223 p.
[7] Pranevicius, H., G. Budnikas. Creation of Estelle/Ag Specifications Using Knowledge Bases. *Informatica*, Vol.14, No.1, 2003: 63-74.
[8] Pranevicius, H., G. Budnikas. Static Verification of Aggregate Specifications. *Information technology and control*, No.4(29), 2003, 39-44.
[9] Pranevicius, H., R. Miseviciene, V.Miliute. Use of Aggregate Specification and Logic Programming for Knowledge Base Validation and Verification. In *Proc. of International Conference* "*Modelling and Simulation of Business Systems*". Technologija, 2003: 203-208.
[10] Pranevicius, H., V. Pilkauskas, A. Chmieliauskas. *Aggregate Approach for Specification and Analysis of Computer Network Protocols*. Technologija, Kaunas. 1994: 254 p.
[11] Preece, A., C. Grossner, T. Radhakrishnan. Validating Dynamic Properties of Rule-Based Systems. *International Journal of Human-Computer Studies*, **44**, 1996: 145-169.
[12] Preece, A., R. Shinghal. Foundation and Application of Knowledge Base Verification. *International Journal of Intelligent Systems*, **9**, 1994: 683-702.
[13] Russel, S., P. Norvig. *Artificial Intelligence—a Modern Approach*. Prentice Hall, 1995.
[14] Sellini, F., C. Vargas, P.-A. Yvars. Considerations about validation of knowledge models in KBE systems. In *Proc. 4th European Symposium on the Validation and Verification of Knowledge Based Systems*. Katholieke Universiteit Leuven, 1997: 83-93.
[15] Specht, G. O!-LOLA - Extending the Deductive Database System LOLA by Object-Oriented Logic Programming. *Informatica*, Vol.9, No.1, 1998: 107-118.
[16] Vanthienen, J. *Prologa v.5 User's manual*, Katholieke Universiteit Leuven, 2000: 121 pp.
[17] Vanthienen, J., C. Mues, A. Aerts. An illustration of verification and validation in the modelling phase of KBS development. *Data & Knowledge Engineering* 27, 1998: 337-352.
[18] Vanthienen, J., C. Mues, G. Wets. Inter-tabular Verification in an Interactive Environment. In *Proc. 4th European Symposium on the Validation and Verification of Knowledge Based Systems*. Katholieke Universiteit Leuven, 1997: 155-165.

SCIENTIFIC PAPERS UNIVERSITY OF LATVIA. 2004. VOLUME 672, Datbases and Information Systems
321–336 P.

A Taxonomy of Characteristics to evaluate specification languages

Albertas Caplinskas, Jelena Gasperovic

Institute of Mathematics and Informatics, IMCS
Akademijos 4, 2600 Vilnius, Lithuania
alcapl@ktl.mii.lt, j.gasperovic@algoritmusistemos.lt

Abstract. This paper presents a quality framework for evaluation of specification languages. It reviews some related works and discusses in detail the components of the proposed framework: quality model, evaluation procedure, and taxonomy of quality characteristics. The main contribution of the paper is the taxonomy and definitions of quality characteristics.

Keywords. IS specification languages, quality models, quality characteristics, quality evaluation

1.Introduction

The influence of specification language on the quality of IS specification is obvious. This problem has been investigated by a number of researchers [9], [10], [12], [13], [18], [20], [21], [22], [23]. However, theoretically well-grounded criteria for specification language evaluation still do not exist. Usually users of a particular language appreciate this language subjectively and express different opinions about its advantages. So the selection of the most appropriate language for the particular project still is a problem. Those are the main reasons why it is very important to propose well-grounded specification language quality evaluation criteria and methodology.

In this paper specification language is understood as a high abstract level language with syntax and semantics appropriate enough to define both the functionality of a system under consideration and its non-functional properties. In general case, the system under consideration can be some real-word business system, information system, software system, hardware system or any other system. We consider only languages that are intended to specify systems, which are relevant to the field of information systems engineering, namely, real-world systems, information systems and software systems. The specifications may be external (requirements specification) or internal (design specification). A specification language that supports the specification process throughout many phases of a life-cycle model is called a wide-spectrum language and a language that supports only one or two phases of a life-cycle model is known as a narrow-spectrum language. We use the term “specification language” in the broad sense that covers both wide-spectrum and narrow-spectrum languages and suppose that wide-spectrum should support business modelling as well as requirement specification and design.

The main objective of this paper is to propose a framework to evaluate IS specification language taking into account high-level quality requirements of a particular project. The paper demonstrates reasonability to describe interdependencies between requirements using special graphs proposed in [3]. It discusses the main ideas beyond the proposed framework and describes in detail its components: quality model, evaluation procedure, and taxonomy of quality characteristics. The paper elaborates ideas proposed in [3]. Its main contribution is the taxonomy and definitions of quality characteristics.

The rest of the paper is organised as follows. Section 2 surveys related works. Section 3 describes the main features of quality model. Section 4 contains description of quality evaluation procedure. Sections 5, 6 and 7 describe the main components of quality model: quality goals, quality assessment function, and quality characteristics tree. Finally, Section 8 concludes the paper.

2. Related works

Firsts attempts to evaluate the quality of specification languages, mostly diagrammatical languages, have been made already in early days of computing era [15]. From nowadays perspective these attempts look rather naïve, however, historically they were very important because initiated entirely new research field. First serious research in this field has been done by Wand and Weber [20], [21], [22], [23]. Roughly speaking, the main idea of this research was to evaluate IS specification languages on the basis of collection of models, known as BWW models, based on an ontology defined by the philosopher Mario Bunge. Quality of specification language is evaluated comparing constructs of this language to some collection of "standard" constructs. This methodology allows evaluating ontological completeness and ontological clarity of a language where ontological completeness is understood as the ability of this language to represent all phenomena of interest in the domain of discourse and ontological clarity is understood as the correspondence between the constructs of a language and the constructs defined by the BWW models. The methodology has been improved by Opdahl [13], who proposed how to evaluate what semantic categories language is able to express and how convenient is it to do.

To the same research direction belongs the works of Milton, Kazmierczak and Keen [11] and of Mylopoulos [12]. The former proposed instead of Bunge ontology to use Chisholm ontology and argued that the quality of a language should be evaluated not from the viewpoint of its constructs but from the point of its ability to specify a variety of situations. The latter argued to evaluate quality in terms of three orthogonal dimensions – ontologies, abstraction mechanisms, and tools – and proposed some qualitative metrics and quality evaluation criteria.

Two other research directions are represented by works of Jackson [7], and Sølvberg and his co-authors [9], [10], [17], [18]. The main idea of Jackson is to use attestation techniques or, in other words, he suggests using a library of characteristic situations that should be specified to evaluate specification language quality. Sølvberg and co-authors proposed to use evaluation framework that addresses quality of specification as well as the quality of specifying process. This framework

includes a language quality model represented in a form of quality characteristics tree. The framework provides a systematic structure for evaluating specification languages. Most important ideas beyond this methodology are separation of conceptual and representation issues, quality evaluation from domain, audience and technology points of view, and the use of set-theoretical approach to explain the meaning of the quality characteristics. It is very advanced approach, but it focuses rather on the quality of specification itself than on the quality of specification languages and investigates the language quality model only occasionally. The proposed language quality model is only sketched. It is not exhaustive and not homogeneous.

There are also a number of works that do not address the quality of specification languages directly, but has a strong impact on the research in this field. First of all, ISO/IEC 9126 standard [6] should be mentioned among them. Although this standard addresses the quality of software its conceptual basis is significantly wider and can be applied to many other fields. For example, the ISO/IEC 9126 standard has been taken as a baseline for QStudio®1 [16] which specifies quality concepts for Java®2 language. This approach defines so-called *Quality Attribute Tree*, which indeed is ISO/IEC 9126 quality model extended by additional sub-characteristics. The *Quality Attribute Tree* is not a proper tree because many of quality sub-characteristics at once refine several characteristics. The "many to many" relationship introduces no difficulties in using the “tree” for quality assessment, because the quality is assessed top down by evaluating the characteristics first and then the sub-characteristics and metrics. So, using stepwise refinement techniques, the notion of code quality is expressed in terms of quality sub-characteristics and mapped further onto programming constructs. In this way quality metrics are attached to the sub-characteristics. The advantage of such approach is the possibility to assess by measurement the quality on multiple levels of detail. Another example is EAGLES/ISO methodology [8] that aims to evaluate the quality of natural language processing systems. It supposes that evaluation expresses what some object is worth to somebody. Quality model is constructed according to two different perspectives. The first perspective (object-based perspective) is who likes it. The second perspective (user-based perspective) is what they like. EAGLES augments the ISO/IEC 9126 approach in the sense that it deals with the formulation of stated or implied needs, which are the primary input to the quality requirement definition. The project aims at producing an evaluation package from which different elements can be taken and combined in different ways to reflect the needs of any particular user. In EAGLES, quality requirements definition is based on the union of the implied needs of classes of users, appropriate metrics are selected and measurements are carried out, but any user is left to construct his own rating level definition and assessment criteria definition [8]. ISO/IEC 9126 also has been used as a basis to develop quality characteristics trees for software components [19] and for ERP systems [2].

One more important approach, called the fuzzy model for software quality evaluation (FMSQE), has been proposed by Belchior and developed further by his colleagues [1]. They proposed a hierarchical quality model based on four main

1 QStudio is a registered trademark of QA Systems BV, The Netherlands .

2 Java is a registered trademark of Sun Microsystems.

concepts: goals, factors, criteria and evaluation processes. Goals represent the general properties that a product should possess. Goals are decomposed in factors; factors can be further decomposed in sub-factors. Factors and sub-factors define different users' perspectives about the quality. Factors (sub-factors) should be decomposed in measurable quality characteristics called criteria. For each criterion one or more alternative evaluation processes, describing a measurement methodology, should be established. In order to obtain the values of factors, both numerical and qualitative measurement results must be aggregated. Measures and aggregate measures are related by quantitative relations. Obtained measures are interpreted using a set of fuzzy functions. Fuzzy functions support the aggregation of measures expressed in different units and are used as a suitable interpretation mechanism able to deal, at the same time, with qualitative measures and numerical data. The proposed approach provides: a membership function mapping the desired quality criterion; a method to calculate the membership function for the aggregate quality; and a final membership function for the whole product.

3. Quality model

There exists no comprehensive definition of quality. Quality ever depends on context. Quality of IS specification language is also relative. It depends on the requirements of a particular project. Following the ISO 8402 [5] definition of quality, the quality of an IS specification language can be defined as the totality of features and characteristics of this language that bear on its ability to satisfy stated or implied project's needs. The language that is excellent for one project may be only acceptable or even unacceptable for some other projects because each project has its specific priorities. Usually the definition of the term "quality" is formalised by a quality model. Quality models are used in many areas, however, often these models are defined imprecisely, only in the form of a quality characteristics tree and, may be, associated metrics. Although some approaches (e.g., ISO/IEC 9126 [6]) address quality in use, quality requirements usually are not seen as a part of quality model. They should be defined separately in terms of quality characteristics tree. It is embarrassing for users because such requirements are low-level requirements even in the case when they are formulated using abstraction levels provided by quality characteristics tree. In [3], we proposed the main idea how to define context-oriented quality model that includes quality requirements and allows formulating those requirements in the form of high-level quality goals. In this paper we elaborate this idea further. We define context-oriented quality model as the following seven-tuple:

$$Q=< \tilde{u}, Ц, Ũ, ф, M, \mu, \xi > (1), \text{ where}$$

$\tilde{u}=<V, A, G, f>$ - a weighted AND/OR digraph describing quality goal interdependencies, termed as "goal graph":

$V=\{v_i \mid 1\le i\le N\}$ – a nonempty set of graph vertices;

$A\subset VxV$ – a nonempty set of ordered pairs of vertices (graph's arrows);

$G=\{g_k \mid 1\le k\le N_1\}$ – a nonempty set of weights;

$f: A\Rightarrow G$ – weighting function that maps graph's arrows to weights;

$l: A\Rightarrow\{AND, OR\}$ – labelling function that marks vertices with the labels "AND" or "OR";

Q- a quality characteristics tree3 with the set of leaf nodes Q_1;

$\varphi = \{\delta_i | \ 1 \le i \le N_2\}$ - a nonempty set of rating levels;

Ū - quality assessment function used to evaluate quality goals that is defined as

$$\Theta(v) = \begin{cases} \sum_{i=1}^{N_3} \delta_i * g_i, & \text{if } v \text{ has the label "AND"} \\ \max_{1 \le i \le N_3} (\delta_i * g_i), & \text{if } v \text{ has the label "OR"} \end{cases} \quad (2), \text{ where}$$

$v \in V$, δ_i – evaluation values that sign vertices adjacent to the vertex v, $g_i \in G$ – weights that sign incoming arrows for the vertex v; N_3 – number of incoming arrows for the vertex v (goals of the lowest levels are evaluated on the basis of rating levels of appropriate quality characteristics);

$M=\{m_i | \ 1 \le i \le N_4\}$ - a nonempty set of quality metrics;

μ: $Q_1 \Rightarrow M$ - one-to-many mapping that relates quality metrics to quality characteristics;

ξ: $Q_2 \Rightarrow \varphi$ - rating function that maps the set of measured values (scores) $Q_2 = \{m(\psi) | \ m \in M, \psi \in Q_1, \mu(\psi)=m\}$ to rating levels.

A quality model for the particular project is designed in the following way. First of all quality goals and their weights should be defined and goals interdependence graph should be developed. Further rating levels should be selected. In the next step, quality characteristics that impact quality goals should be selected and the most appropriate metrics for measuring these characteristics should be chosen. Finally, quality assessment criteria should be developed. Note that quality characteristics tree Q defines internal quality of the language or, in other words, it does not depend on the particular project and, consequently, is the same for any quality model. However, not all quality characteristics might be important from the viewpoint of the particular project and in the particular quality model some characteristics might be missed.

Let us consider the example of a simple quality model Q1 designed for project P1 in order to choose the most appropriate IS specification language.

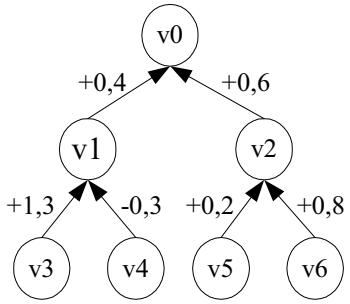


Figure 1. The weighted digraph describing interdependencies between quality goals

Example 1.

1. Goal interdependencies graph ũ is presented in Fig.1.

3 Following tradition, we use the term "quality characteristics tree", although, it is a hierarchical classification of quality characteristics, represented by tree.

Weights set is defined as G={-0.3, +0.2, +0.4, +0.6, +0.8, +1.3}.

Weighting function f is defined as follows:

f(v3,v1)=+1.3; f(v4,v1)=-0.3; f(v1,v0)=+0.4;

f(v5,v2)=+0.2; f(v6,v2)=+0.8; f(v2,v0)=+0.6;

2. Quality characteristics tree Q is presented in Fig. 2.

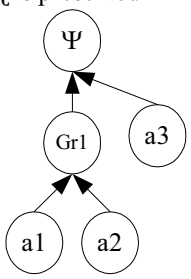


Figure 2. Quality characteristics tree

The set of tree leaf nodes is defined as Q_l= {a1, a2, a3}.

3. The set of rating levels is defined as φ = {1 (unacceptable), 2 (acceptable), 3 (good)};

4. Quality assessment function Ū is defined as follows:

Ū(v1)= 1.3 Ū(v3)-0.3 Ū(v4);

Ū(v2)= 0.2 Ū(v5)+0.8 Ū(v6);

Ū(v0)= 0.4 Ū(v1)+0.6 Ū(v2);

Ū(v3)= 1.5 ξ(a1) – 0.5 ξ(a2);

Ū(v4)= 0.2 ξ(a2) + 0.8 ξ(a3);

Ū(v5)= 0.3 ξ(a1) + 0.8 ξ(a2) – 0.1 ξ(a3);

Ū(v6)= 0.8 ξ(a1) + 0.2 ξ(a3);

5. The set of quality metrics is defined as μ ={m1, m2}. Metrics are related to quality characteristics in the following way:

μ={(m1,a1), (m2,a2), (m2,a3)}

6. The rating function ξ is defined as follows:

$$\xi(m1(a1)) = \begin{cases} 1, \text{if } 0 \le m1(a1) < 1, \\ 2, \text{if } 1 \le m1(a1) < 5, \\ 3, \text{if } 5 \le m1(a1) \end{cases}$$

$$\xi(m2(a2)) = \begin{cases} 1, m2(a2) < 15, \\ 2, \text{if } 15 \le m2(a2) < 20, \\ 3, \text{if } 20 \le m2(a2) \end{cases}$$

$$\xi(m2(a3)) = \begin{cases} 1, \text{if } m2(a3) < 25, \\ 2, \text{if } 25 \le m2(a3) < 30, \\ 3, \text{if } 30 \le m2(a3) \end{cases}$$

4. Evaluation procedure

The proposed user-oriented quality model suggests appropriate quality evaluation procedure. This procedure includes four steps:

1. The value of every quality characteristic from tree Q is measured using metrics m_i from the set M.
2. Rating level of every measured quality characteristic value is determined by the values of rating function ξ arguments.
3. The rating levels of terminal quality goals are defined using quality assessment function Ū.
4. The rating levels are propagated through the goal interdependencies graph ũ using quality assessment function Ū.

Example 2.

The evaluation of IS specification language L1 according to quality evaluation procedure provided by the quality model Q1 presented in *Example 1* gives the following results:

1. m1(a1)=3; m2(a2)=15; m2(a3)=25;
2. ξ(3) = 2; ξ(15) = 2; ξ(25) = 2;
3. Ū(v3)= 1; Ū(v4)= 1; Ū(v5)= 2; Ū(v6)= 2;
4. Ū(v1)= 2; Ū(v2)= 2; Ū(v0)= 2.

Thus, the quality of evaluated specification language L1 for the project P1 is acceptable (rating level 2).

5. Quality goals

In the proposed approach, high-level quality requirements are formulated in the form of quality goals. The user's treatment of quality goals and interdependencies between goals are described using goal interdependency graph (GIG) ũ. The idea of GIG is borrowed from [1], where similar graphs, namely, softgoal interdependency graphs (SIG), are used to define non-functional software requirements. The advantage of GIG, comparing to quality characteristics tree, is that user can start from business goals, formulate high-level quality requirements and derive further detailed quality requirements, formulated in the terms of low-level quality characteristics. For example, top management may aim to spend money on staff training, minimise the amount of efforts needed to produce specifications, and produce specifications readable by domain experts. In other words, management is looking for an IS specification language that would be simple enough to learn in short terms by not skilled staff, would be efficient and would have high degree of audience appropriateness. GIG allows defining goal interdependences and priorities, helps to expose implicit interdependencies and to refine requirements up to low-level quality characteristics. By refinement, for each goal a set of subgoals to satisfy this goal is introduced. Parent goal may be satisfied by all of its subgoals or by any of them. In addition, some subgoals can contribute positively towards a particular goal and negatively towards another goals.

Although GIGs are very similar to SIGs, they are used for different purposes and in different way. The main problem investigated in [4] is software design problem. SIGs are used in top-down manner with the aim to refine non-functional requirements, including quality requirements, to choose design decisions, which accomplish those requirements in the possibly best way, and to evaluate the impact of chosen decisions in bottom-up way. Our task is to choose such IS specification language, which satisfies quality requirements in the best way. So we need to choose not the design decisions but the most appropriate metrics and measurement procedures and, further, to propagate evaluation results through the GIG in the bottom-up manner.

6. Quality assessment function

Each quality goal is evaluated using quality assessment function $\bar{U}$. This function is used to propagate rating levels of quality characteristics through the GIG from bottom towards the top of the graph. Measured values are mapped to rating levels because, in a general case, they are incomparable, expressed in different dimensions. Quality assessment function of the lowest GIG levels is calculated using measured values of quality characteristics from tree Ψ. These values are used as intermediate to calculate quality assessment function of the highest GIG level. Quality assessment function depends on goals priorities and, consequently, is designed in such way that to evaluate the impact of quality characteristics of the chosen language on the quality goals. Because the arrows are signed by negative and positive priorities, the calculated value may be not exactly integer. It means that we propagate towards top of the graph not the rating levels itself but some values that belong to intervals [v1, v2], where v1 and v2 are adjacent rating levels.

For “AND” vertices the value of quality assessment function is calculated taking into account the weights of all incoming arcs. In this case, positive as well as negative weights are allowed, and it is required that the sum of weights should be equal to 1. For “OR” vertices the value of the $\bar{U}$ is the maximal one from the propagated values. In this case, only positive weights are allowed, and it is required that any weight do not exceed 1.

7. Quality characteristics

A number of publications on IS specification problems [9], [10], [13], [17], [18], [20], [21], [22], [23] mention different quality characteristics that specification language should have. Some quality characteristics discussed in publications on the quality of programming languages and cognitive dimensions of information artefacts (see, for example, [14]) also are closely related to the quality of specification languages. However, exhaustive set of quality characteristics up to date still has not been proposed. Extensive library research [3] has brought to light that some important quality characteristics even have no names and some important groups of quality characteristics are not thought as a coherent groups. Also in the field of programming languages, where research has already a long history, different

researchers evaluate quality of programming languages using slightly different quality characteristics and classify these characteristics in different ways. For example, some researchers argue that reliability of a language is influenced by both readability and writability, others – that it is influenced only by readability. In addition, the characteristics of the internal quality and characteristics of the quality in use often are not entirely separated. On the basis of the results of conceptual analysis of wide spectrum of specification languages, including UML, Z, VDL, Troll, and Alloy, we decided to organise quality characteristics of specification language in a way proposed by ISO/IEC 9126 standard [6]. From six groups of software quality characteristics (functionality, reliability, usability, efficiency, maintainability, and portability) provided by this standard we have left only four because maintainability and portability are relevant rather to software tools supporting production of specifications than to specification language itself. The current version of the quality characteristics tree is presented in Appendix 1. Let us comment this tree shortly.

Following the idea, proposed by Krogstie and Sølvberg [9] and elaborated in [3], we describe quality characteristics in terms of linguistic system, defining a formal structure beyond the language, and in terms of representation system, defining forms of representation of its constructs. Further, we treat the concepts of functionality, reliability, usability, and efficiency in a different way than ISO/IEC 9126 standard and, consequently, refine these characteristics using other lower level sub-characteristics. Functionality of a specification language is understood as the existence of means, required to specify functional and non-functional properties of the system under consideration. Functionality has two dimensions: suitability and flexibility. The first one characterises the kind of systems that can be described using a given language, the second one – the spectrum of describable systems (real-world systems, IS, software systems, etc.). We suppose that the specification of the system can be seen as a collection of statements about the properties of this system. Statements can be explicit, or implicit, derivable from other statements. The possibility to produce such specification first of all depends on the completeness of the language or, in other words, on the ability to express all necessary kinds of knowledge at all representation levels: epistemological, ontological, and conceptual [3]. We present only first two levels of the completeness characteristics. This is far not enough to operationalise completeness. At least three or four additional levels should be added for this aim.

It should also be noted that we define two unusual characteristics: paradigm and ontology. Although these characteristics describe important properties of a language, they are not related directly to the internal quality of this language. However, evaluation of the quality in use often requires considering formalism beyond specification language and assessing of ontological aspects of this language. Besides, these characteristics are not entirely descriptive and sometimes should be operationalised, for example, a metric should be developed to measure the degree of procedureness of the language.

8. Conclusions and further research

Despite the quality of specification languages has strong impact on the quality of specifications research in this field is still in beginning. Although specification languages in many aspects are similar to programming languages, quality models of the latter cannot be applied directly to the former. Additionally, quality models of programming languages mostly also are only sketched. No commonly accepted agreement exists about the collection of quality characteristics, their names and their taxonomy, internal quality is not separated from quality in use.

This paper continues research on the evaluation of the quality of specification languages, which has begun in [3]. It should be considered as a first attempt to develop exhaustive quality characteristics tree to evaluate internal quality and procedure to evaluate quality in use. Further research provides elaboration and operationalisation of the quality characteristics tree, development of the quality characteristics tree for the quality in use and, maybe, improvement of the evaluation procedure.

References

[1] Belchior, A. D., Xexéo, G., da Rocha, A. R. C. Evaluating Software Quality Requirements using Fuzzy Theory. In: Proceedings of ISAS 96, Orlando, July, 1996, http://www.cos.ufrj.br/~xexeo/artigos/isas96/isas96.htm

[2] Botella, P., Burgués, X., Carvallo, J.P., Franch, X., Pastor, J.A., Quer, C. Towards a Quality Model for the Selection of ERP Systems. In Cechich, A., Piattini, M., Vallecillo, A. (eds.). Component-Based Software Quality: Methods and Techniques. LNCS 2693, Springer, 1998, 225-245.

[3] Caplinskas, A., Lupeikiene, A., Vasilecas, O. A Framework to Analyse and Evaluate Information Systems Specification Languages. In Manolopoulos, Y., Navrat, P. (eds.). Advances in Databases and Information Systems. 6th East European Conference, ADBIS 2002, Bratislava, Slovakia, September 2002, Proceedings. LNCS 2435, Springer, 2002, pp. 248-262.

[4] Chung, L., Nixon, B.A., Yu, E., Mylopoulos, J. Non-functional Requirements in Software Engineering. Kluwer Academic Publishers, 2000.

[5] ISO 8402. Quality management and quality assurance vocabulary. Second edition. (1994-04-01).

[6] ISO/IEC 9126. Information Technology – Software Product Evaluation – Quality Characteristics and Guidelines for their Use. First edition, 1991-12-15, reference number ISO/IEC 9126: 1991(E).

[7] Jackson, D. A Comparison of Object Modelling Notations: Alloy, UML and Z. Unpublished manuscript. MIT Lab for Computer Science, August 1999, retrieved January 10, 2004 from http://geyer.lcs.mit.edu/~dnj/pubs/alloy~comparison.pdf

[8] King, M., Maegaard, B. Issues in Natural Language Systems Evaluation. In: Proceedings of the First Conference on Language Resources and Evaluation, Granada 1998, 225-230, retrieved January 10, 2004 from http://citeseer.nj.nec.com/514907.html.

[9] Krogstie, J., Sølvberg, A. Information Systems Engineering: Conceptual Modeling in a Quality Perspective. The Norwegian University of Science and Technology, Andersen Consulting (January 2, 2000). The draft of the book.

[10] Lindland, O.I., Sindre, G., Sølvberg, A. Understanding Quality in Conceptual modelling. IEEE Software (March 1994) 42-49.

[11] Milton, S., Kazmierczak, E., Keen, C. Comparing Data Modelling Frameworks Using Chisholm's Ontology, In Proceedings of the 4th European Conference on Information Systems, ECIS'98, Aix-en-Provence, June 1998, 1998.
[12] Mylopoulos, J. Characterizing Information Modeling Techniques. Bernus, P., Mertins, K., Schmidt, G. (eds.). Handbook on Architectures of Information Systems. Springer, Berlin, 1998, pp. 17-57.
[13] Opdahl, A. L**.** Applying Semantic Quality Criteria to Multi-Perspective Problem Analysis Methods. In Dubois, E., Opdahl, A. L., Pohl, K. (eds.). Proceedings of "The Third International Workshop on Requirements Engineering: Foundations of Software Quality — REFSQ'97", Barcelona/Spain, June 1997, 1997.
[14] Pane, J.F., Myers, B. A. Usability Issues in the Design of Novice Programming Systems. Technical Report CMU-CS-96-132, School of Computer Science, Carnegie Mellon University, Pittsburgh, 1996, retrieved January 10, 2004 from http://www-2.cs.cmu.edu/~pane/cmu-cs-96-132.html.
[15] Peters, L.J., Trip, L.L.: Comparing Software Design Methodologies. Datamation, 23(11), (1977) 89-94.
[16] QStudio® for Java. The Software Health Tool for Java. QA Systems BV, The Netherlands, 2003, retrieved January 10, 2004 from http://www.qa-systems.com/welcome.html.
[17] Seltveit, A.H.: Complexity Reduction in Information Systems Modelling. PhD thesis, IDT, NTH, Trondheim, Norway (1994).
[18] Sindre, G. HICONS: A General Diagrammatic Framework for Hierarchical Modelling. PhD thesis, IDT, NTH, Trondheim, Norway (1990).
[19] Simão R. P.S., Belchior A.D. Quality Characteristics for Software Components: Hierarchy and Quality Guides. In Cechich, A., Piattini, M., Vallecillo, A. (eds.). Component-Based Software Quality: Methods and Techniques. LNCS 2693, Springer, 1998, 184-206.
[20] Wand, Y., Weber, R. An ontological analysis of systems analysis and design methods. In Falkenberg, E., Lindgreen, P. (eds.). Information Systems Concepts 95— An In-Depth Analysis, North-Holland, 1989, 79–107.
[21] Wand, Y., Weber, R. An Ontological Evaluation of Systems Analysis and Design Methods. In: Falkenberg, E.D., Lindgreen, P. (eds.): Information Systems Concepts: An In-depth Analysis. North-Holland, Amsterdam (1989) 79-107.
[22] Wand, Y., Weber, R. On the Ontological Expressiveness of Information Systems Analysis and Design Grammars. Journal of Information Systems 3(4) (1993) 217-237.
[23] Wand, Y., Weber, R. On the Deep Structure of Information Systems. Information Systems Journal, 1995, 5, 203-223.

Appendix 1. – Specification language quality characteristics definitions

1	**Functionality**	Characteristics of a linguistic system that bear on the existence of means required specifying a given phenomenon.
1.1	Suitability	Characteristics of a linguistic system that bear on its conceptual appropriateness.
1.1.1	Completeness	Characteristic of a linguistic system that bears on its ability to express statements about a given phenomenon.
1.1.1.1	Semantic adequacy	Characteristics of a linguistic system that bear on its ability to express semantic primitives.
1.1.1.1.1	Epistemological adequacy	Characteristic of a linguistic system that bears on its ability to express epistemological primitives.
1.1.1.1.2	Ontological adequacy	Characteristic of a linguistic system that bears on its ability to express its ontological commitments.
1.1.1.1.3	Conceptual adequacy	Characteristic of a linguistic system that bears on its ability to express cognitive primitives.
1.1.1.2	Composability	Characteristic of a linguistic system that bears on its ability to express compositions of semantic primitives.
1.1.1.3	Expressibility	Characteristic of a linguistic system that bears on its ability to express statements about properties of semantic primitives and their compositions.
1.1.1.4	Reasoning power	Characteristic of a linguistic system that bears on its ability to derive new statements about properties of semantic primitives and their compositions.
1.1.2	Expressive adequacy	Characteristics of a linguistic system that bears on its ability to express semantic primitives with the needed degree of precision.
1.1.2.1	Selective power	Characteristic of a linguistic system that bears on its ability to distinguish

		details with the needed degree of precision.
1.1.2.2	Generalitive power	Characteristic of a linguistic system that bears on its ability to support abstractions and to hide details.
1.1.3	Paradigm	Characteristic of a linguistic system that describes mathematical theory beyond the language.
1.2	Flexibility	Characteristics of a linguistic system that bears on the spectrum of its applicability (real-world system, IS, software system, etc.).
1.2.1	Adaptability	Characteristic of a linguistic system that bears on its ability to specialise its constructs.
1.2.2	Universality	Characteristic of a linguistic system that bears on its ability to apply its constructs to a wide spectrum of phenomena.
1.2.3	Extensionability	Characteristic of a language (both linguistic system and representation system) that bears on its ability to define new constructs.
2	**Reliability**	Characteristics of a linguistic system that bear on its ability to ensure correctness of specifications.
2.1	Error robustness	Characteristic of a language (both linguistic system and representation system) that bears on the number of errors in the specification.
2.1.1	Precision of semantics	Characteristic of a linguistic system that bears on unambiguity of the concepts of language.
2.1.2	Precision of notation	Characteristic of a representation system that bears on unambiguity of the structure of language constructs.
2.1.3	Distinguishibility	Characteristic of a representation system that bears on absence of subtle distinctions in syntax, which might be overlooked or confused.
2.1.4	Simplicity	Characteristic of a language (both linguistic system and representation system) that bears on the comprehensibility of language
2.1.4.1	Conceptual simplicity	Characteristic of a linguistic system that bears on the comprehensibility (including conceptual cleanliness) of the concepts defined by language.

2.1.4.2	Functional simplicity	Characteristic of a linguistic system that bears on a number of features of language.
2.1.4.3	Representational simplicity	Characteristic of a representation system that bears on the comprehensibility (including syntactical transparency) of the constructions of language.
2.1.5	Semantic power	Characteristic of a linguistic system (level of language) that bears on the semantic power of the concepts defined by this system.
2.1.6	Orthogonality	Characteristic of a linguistic system that bears on the degree in which the concepts of the language interfere with each other.
2.1.7	Uniformity	Characteristic of a language (both linguistic system and representation system) that bears on the degree of internal standardisation of the syntax (syntactical uniformity) and the semantic (semantic uniformity) of the construction of the language.
2.2	Verifiability	Characteristic of a linguistic system that bears on the ability (including executability) to check correctness of produced specifications.
3	**Efficiency**	Characteristics of a linguistic system and a representation system that bear on the amount of efforts needed to produce specification.
3.1	Expressive efficiency	Characteristic of a linguistic system that bears on the efforts necessary to express the statements about the phenomena.
3.2	Representational efficiency	Characteristic of a representation system that bears on efforts necessary to represent the statements about the phenomena.
3.3	Semantic power	Characteristic of a linguistic system (level of language) that bears on the semantic power of the concepts defined by this system.
3.4	Orthogonality	Characteristic of a linguistic system that bears on the degree in which the concepts of the language interfere with each other.
3.5	Permissiveness	Characteristic of a language (both

		linguistic system and representation system) that bears on its ability to express and represent things in several different ways.
3.6	Viscosity	Characteristic of a language (both linguistic system and representation system) that bears on the possibility to narrow the change impact on the produced specification.
3.7	Technological efficiency	Characteristics of a language (both linguistic system and representation system) that bear on the technological appropriateness of a language.
3.7.1	Manageability	Characteristics of a language (both linguistic system and representation system) that bear on appropriateness of the language to deal with large and complex specifications.
3.7.1.1	Decomposability	Characteristic of a language (both linguistic system and representation system) that bears on the possibility to divide the produced specification into relatively autonomous and independent subparts.
3.7.1.2	Genericity	Characteristic of a linguistic system that bears on the possibility to use generic types.
3.7.2	Processability	Characteristic of a language (both linguistic system and representation system) that bear on appropriateness of the language to manipulate specifications by software.
4	**Usability**	Characteristics of a linguistic system and a representation system that bear on the audiences' effort to understand and to learn a language.
4.1	Understandability	Characteristics of a linguistic system that bear on the audiences' effort to understand its conceptualisation.
4.1.1	Ontology	Characteristic of a linguistic system that describes language ontology.
4.1.2	Naturalness	Characteristic of a linguistic system that bears on the correspondence between the language ontology and the common sense.
4.1.3	Precision of semantics	See 2.1.1
4.1.4	Precision of notation	See 2.1.2

4.1.5	Distinguishibility	See 2.1.3
4.1.6	Uniformity	See 2.1.7
4.2	Learnability	Characteristics of a linguistic system and a representation system that bear on the audiences' effort for learning of language application.
4.2.1	Simplicity	See 2.1.4
4.2.2	Semantic power	See 3.3
4.2.3	Naturalness	See 4.1.2
4.2.4	Uniformity	See 2.1.7
4.2.5	Commonality	Characteristic of a language (both linguistic system and representation system) that make the language adhere to wide-accepted standards or conventions.

MODEL TRANSFORMATIONS

SCIENTIFIC PAPERS UNIVERSITY OF LATVIA. 2004. VOLUME 672, Datbases and Information Systems
339–357 P.

Model Transformation Language MOLA: Extended Patterns

Audris Kalnins, Janis Barzdins, Edgars Celms

University of Latvia, IMCS
29 Raina boulevard, Riga, Latvia
{Audris.Kalnins, Edgars.Celms}@mii.lu.lv

Abstract. The paper describes a new graphical transformation language MOLA for MDA-related model transformations. Transformations in MOLA are described by combining traditional control structures, especially loops, with pattern-based transformation rules. Besides an overview of the basic MOLA, the paper describes an extension of MOLA – powerful patterns, which may include transitive closure. The paper shows how the usage of these patterns simplifies control structures for typical MDA tasks.

Keywords. Model transformations, MDA, patterns

1. Introduction

The increased use of modeling techniques nowadays requires effective support for model transformations. Perhaps, one of the most actual areas in software engineering today is the Model Driven Architecture (MDA) [1]. MDA is a software development approach in which models are the primary artifacts. According to the MDA, the different types of models are defined (most usually in UML [2,3] notation), mapping between models and towards different targets is formalized, and guidelines are automated, in order to improve efficiency and guarantee that the process of the transformation between models is properly followed. Model-to-model transformation is therefore a key technology for MDA. While the current Object Management Group (OMG) standards such as the Meta Object Facility (MOF) [4] and the UML provide a well-established foundation for defining different types of models, no such well-established foundation exists for transformations between them. The need for standardization in this area led to the MOF 2.0 Query/Views/Transformations (QVT) request for Proposals (RFP)[5] from OMG.

To a great degree the success of the MDA initiative and of QVT in particular will depend on the availability of a concrete syntax for model-to-model transformations that is able to express non-trivial transformations in a clear and compact format that would be useful for industrial production of business software [6].

The submissions by several consortiums have been already made, e.g. [7, 8, 9], and it is somewhat surprisingly, that only a few of them use a natural graphical representation of their language. Currently none of them has reached the status of a complete model transformation language. Several proposals for transformation

languages have been provided outside the OMG activities. The most interesting and complete of them seem to be UMLX [10] and GReAT [11].

According to our view, and many others [6], model transformations should be defined graphically, but combining the graphical form with text where appropriate. Graphical forms of transformations have the advantage of being able to represent mappings between patterns in source and target models in a direct way. This is the motivation behind visual languages such as UMLX, GReAT and the others proposed in the QVT submissions. Unfortunately, the currently proposed visual notations do not provide easy readable descriptions of model transformations.

The common setting for all transformation languages is such that the model to be transformed – the source model is supplied as set of class and association instances conforming to the source metamodel. The result of transformation is the set of instances conforming to the target metamodel – the target model. Therefore the transformation has to operate with instance sets specified by a class diagram (actually, the subset of class notation, which is supported by MOF).

Approaches that use graphical notation of model transformations draw on the theoretical work on graph transformations. Hence it follows that most of these transformation languages define transformations as sets of related rules. Each rule consists of a pattern and action part, where the pattern has to been found (matched) in the existing instance set and the actions specify the modifications related to the matched subset of instances. This schema is used in all of abovementioned graphical transformation languages. Languages really differ in the strength of pattern definition mechanisms and control structures governing the execution order of rules.

The most detailed pattern definition is in the GReAT language. There it is possible to match a set of instances to one element of the pattern (variable cardinality patterns). However, the patterns are still limited in depth but this is compensated by a very elaborated rule control structure specified graphically by dataflow-like diagrams. UMLX has a similar but slightly weaker pattern mechanism. The control structure is completely based on recursive invocations of rules. In the proposal by QvT-Partners [7] graphical patterns are combined with extensive use of textual constraints. The control structure is based on recursive invocation of rules. In the DSTC/IBM/CBOP proposal [12] (now merged with [7]) patterns are specified in a textual (Prolog-like) form, the most interesting feature of this language is the possibility to include a transitive closure in patterns.

This paper proposes a new graphical transformation language MOLA (Model Transformation Language). The main design goal for MOLA has been to make the transformation definitions natural and easy readable, by relying on simple iterative (non-recursive) control structures, based on traditional structured programming. In addition, as far as it improves readability, the intention was to make each rule more powerful. In particular, this requires the strengthening of pattern mechanism. The MOLA project actually consists of two parts – the basic and extended MOLA. The basic MOLA uses simple patterns and more relies on control structures – it has a more procedural style. The main element there is a graphical loop concept, which can easily be combined with a transformation rule. The main new feature of the extended MOLA is the possibility to define looping patterns of “unlimited depth” (in addition to variable cardinality), thus incorporating the mechanism of transitive closure in patterns. In the result, very simple control structure – a sequence of simple loops then is sufficient for many transformation jobs. Certainly, such a pattern

definition requires an adequate definition of the matching procedure, which is also described in the paper. High execution efficiency for the procedure is guaranteed in the typical case when the pattern cardinalities are adapted to the metamodel multiplicities, to which the instance set conforms (the "uniqueness principle" is observed). Patterns in MOLA are defined as directed graphs, to enable the required control over the matching procedure – also a new feature for pattern definition. As a consequence of larger and more powerful patterns, a typical step of a transformation frequently can be described by one rule. This natural non-recursive style of transformation definitions in MOLA has been tested on several real MDA jobs, at the same time using the features of MOLA to keep each rule not too complicated (namely the right balance there ensures the human readability of transformations). As far as we know, the extended MOLA is the only graphical transformation language, which supports transitive closure in patterns. The ideas for pattern definition in MOLA have been partially inspired by the authors' previous experience in defining mappings for generic modeling tools based on metamodels [13].

This paper describes the basic elements of MOLA. The main emphasis is on the extended pattern concept. Language description is based on a typical MDA example (used also in [7, 8, 10, 14]) – transformation of a simplified class diagram into a database definition. Section 2 describes the general structure of MOLA, section 3 - the example. Section 4 describes the basic MOLA – rules with simple patterns and the new concept of loop. The complete pattern mechanism in extended MOLA, including cardinality constraints, looping patterns and the corresponding matching procedure is described in section 5.

2. Overview of language elements in MOLA.

The MOLA language is a natural combination of pattern-based model transformation rules with control structures from traditional structured programming, both specified in a graphical form.

MOLA is meant for transforming models built according to one metamodel – the **source metamodel** (SMM) to models conforming to another metamodel – the **target metamodel** (TMM). In a special case, SMM and TMM may coincide. Both the source and target models actually are treated as instance sets of the corresponding metamodel classes and associations.

A **transformation definition** in MOLA consists of the both metamodels and the transformation program. A **transformation program** in MOLA is a sequence of **statements**. A statement is a graphical area, delimited by a rectangle – in most cases, a gray rounded rectangle. The statement sequence is shown by dashed arrows. The program starts with the UML start symbol and ends with an end symbol.

The simplest kind of statement is a **rule**, which performs an elementary transformation of instances. A rule contains a **pattern** – a set of elements representing class and association instances, built in accordance with the source metamodel. Pattern elements can have **attribute constraints** (OCL expressions). The pattern specifies what kind of instance group must be found in the source model, to which the rule must be applied. A rule has also an **action** specification – new class instances to be built, instances to be deleted, association instances (links)

to be built or deleted and the modified attribute values (as assignments). Both for the pattern and action part the UML object (instance specification) notation is used.

The most important statement type in MOLA is the **loop**. Graphically a loop is a rectangular frame, containing a sequence of statements. This sequence starts with a special **loop head** statement. The loop head is also a pattern, but with one element – the **loop variable** highlighted (by a **bold** frame). Informally a loop variable represents an arbitrary instance of the given class, which satisfies the conditions specified by the pattern. Actually there are two types of loops in MOLA, differing in semantics details. The first type (denoted by a **simple** frame) is executed once for each valid loop variable instance, therefore it is called **FOREACH** loop. Mainly this type of loop will be used in the paper. The second type (denoted by a **3-d** frame) is executed while there is at least one loop variable instance satisfying the pattern conditions – it is called the **WHILE** loop. Loops can be nested to any depth. A loop head can contain actions – it is also a rule. Such a combined statement will be widely used in the examples of this paper.

Other control structures in MOLA are the **branch** construct – several frames started by pattern statements as conditions and the **subprogram** concept together with the **subprogram call**, where the parameters can contain references to instances used in the calling program. However, these control structures actually will not be used in the paper – the aim of this paper is to demonstrate how extended patterns in MOLA allow to use a very simple control structure – a sequence of simple loops.

Section 4 discusses in detail the syntax and semantics of basic MOLA on an example. Then the extended patterns are introduced – the section 5 describes the extended MOLA.

The general execution schema in MOLA is simple – when a source model is supplied, statements are applied to it in the specified order. A statement is always applied to the whole instance set which is being gradually transformed by the rule actions. The potential execution efficiency is ensured by the corresponding features of the pattern language, where it is easy to specify that only “useful” pattern matches occur.

During the transformation process one more optional metamodel – the **intermediate metamodel** (IMM) may be used. IMM contains both SMM and TMM as a subsets, and additional elements – classes, attributes and associations necessary for performing the transformation. There is a special kind of additional associations – **mapping associations** which in fact are present in every transformation. These associations physically implement the mapping from the elements of source model to target elements, therefore they link classes from SMM to the corresponding classes in TMM. See more on the role of mapping associations in 4.2. Namely due to a large set of mapping associations it is recommended to use IMM for non-trivial transformations (it is also permitted in MOLA to define mapping associations “on the fly” – directly in rules, if IMM is not used). Another important elements of IMM are **computed attributes** – “temporary” attributes added to classes of IMM for storing intermediate values. There may be several kinds of computed attributes, the most used here are the rule-local ones. Names of rule-local attributes start with “?” in the IMM, see more on them in 5.2. Non-local temporary attributes (with the scope of several statements) can be also defined/created and destroyed by special statements.

3 Example - the class model to relational model transformation

The paper will be based on a typical MDA example, considered also in [7, 8, 10, 14] – the transformation of a simplified class diagram into a semantically equivalent relational database definition. For all the different versions of the example the one in [7] is used here. This version permits to demonstrate the easy definition of transitive closure in extended MOLA. The SMM for this task is shown in Fig.1.

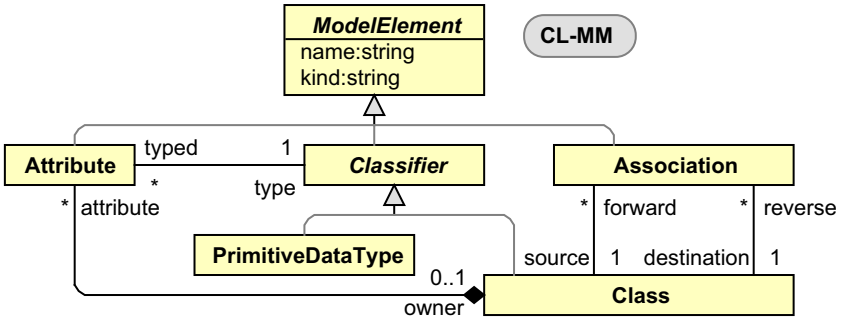


Figure 1. Source metamodel for simplified class diagram.

All elements can have a *name*. The metaattribute *kind* is applicable to metaclasses *Class* and *Attribute*, only a *Class* where its value is "*persistent*" must be transformed into a database *Table*, the value of *kind* equal to "*primary*" determines that an *Attribute* actually is a part of a primary key (all other values of *kind* are irrelevant). The *type* of an *Attribute* can be either a *PrimitiveDataType*, or another *Class*. Fig.2 shows the TMM - a simplified relational database definition metamodel.

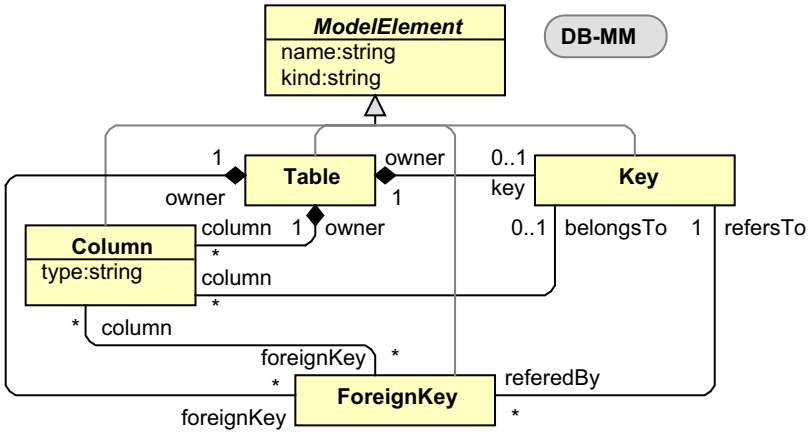


Figure 2. Target metamodel for database definition.

A *Table* consists of *Columns*, and it can have a (primary) *Key*, which contains some of the *Columns*. The *ForeignKey* for a *Table* always refers to a *Key* of another *Table*.

The informal transformation algorithm is quite straightforward. For each *persistent* class a table and its key must be built. The primitive-typed attributes of this class become columns of the table (with the same name and type name). The columns which correspond to primitive-typed attributes of *primary* kind become parts of the key. In addition, for class-typed attributes, the primitive attributes of the target class are also transformed to columns of the table for the original class (as "indirect attributes"), and this process of finding indirect attributes is continued down until no more indirect attributes can be found (so-called class flattening – a transitive closure-like process). A column for an indirect attribute has a compound name – the concatenation of all attribute names down to the primitive one. An association is converted into a foreign key for the source class (table), and this foreign key refers to the key for the destination class. In addition, new columns are added to the source table (and to the foreign key) – one (equally named and typed) for each column of the corresponding key. The problem of ordering the columns in keys is ignored in the example. Association multiplicities also are not used in this simplified example – only the association direction matters. It should be noted that the processing of indirect attributes and associations is independent – it may be done in any order.

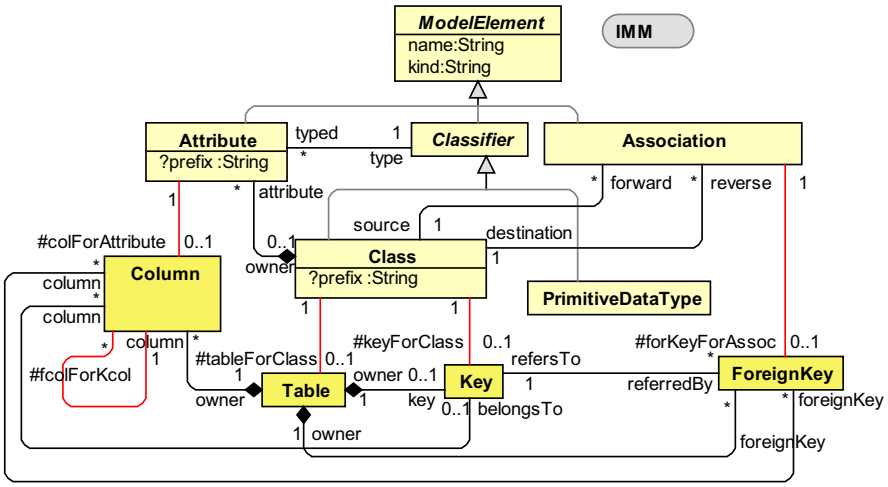


Figure 3. Intermediate metamodel for the transformation.

The one remaining element to be described is the IMM – see Fig.3. In addition to a copy of SMM and a copy of TMM (the classes of TMM are in a darker color), it contains mapping associations from source to target elements, e.g., from *Class* to *Table*. These mapping associations will be used in rules in sections 4 and 5, they have the # character prefixed to role names and are in red color in Fig.3. IMM contains also a computed attribute *prefix* of rule-local kind (names of rule-local attributes start with "?" in MOLA), it will be used in section 5.2.

4. Structure of simple loops and rules

As it was stated, both loops and rules rely on patterns in MOLA. In this section the structure of simplest patterns will be described in detail. These patterns, for which only a fixed-size match is possible, have a very simple matching algorithm. The patterns in this section actually are weaker than those described in [10, 11], the goal of this section is just to demonstrate the general principles of MOLA in a very simple case. Non-trivial patterns of MOLA will be described in section 5.

4.1. Basic patterns

A pattern in MOLA specifies the instance set which can be matched to it. From a syntax point of view, it is similar to UML 2.0 collaborations or structured classifiers. The main **element** of a pattern is a source **metamodel class,** specified in UML **instance notation**. Each element has an optional **instance name** and the **class name,** the same class may be used several times in a pattern. In totality, they must be unique within a pattern. Each element matches to an appropriate instance of that class. Since a typical use of pattern in MOLA is in a loop head, we start with this case. There one pattern element – the loop head (a bold one) has a special meaning. All other elements of the pattern are used to specify, namely which instances of this class in the source model can be used as loop variable instances. The other pattern elements (which may correspond also to target metamodel classes) also must match to an appropriate instance – they specify the context of a loop variable.

In addition to elements, a pattern contains **pattern associations** – selected metamodel associations between the used classes and **attribute constraints** – OCL constraints specified within elements (in braces). The specified association instances must exist between the matched model instances and the attribute constraints must evaluate to true. Pattern associations can have also a {NOT} constraint – this means that no specified instance can be linked to the "main match" by the given link.

Thus a pattern in a loop head specifies which instances of the given class in the source model qualify as valid instances for the loop variable. The other pattern elements have the "**exists**" semantics – there must be (or must not be) an appropriate instance in the selected match, but there is no need to find all possible matches for them.

The loop variable in a pattern in fact plays the role of its **root** – the match is started from it.

Fig.4 shows the simplest pattern consisting just of the loop variable. This pattern says that an instance of *Class* in the source model (i.e., the instance set corresponding to the class diagram to be transformed into a database definition) matches to this pattern, if its *kind* has the value *"persistent"* (*kind* – a string-typed attribute of *Class*).

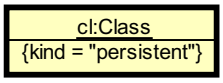


Figure 4. Simplest pattern example.

Fig. 5 shows a more non-trivial pattern, involving several elements and associations. In this pattern only these instances of *Attribute* qualify as a loop variable instances, which have an *owner* link to a Class instance, which in turn has a *#tableForClass* link to a *Table* instance, and also have a *type* link to a *PrimitiveDataType* instance. The *Table* class is from the target metamodel, and the *#tableForClass* is a mapping association – this means that these instances have to be already built by previous statements. Pattern associations typically specify only one of the role names – that leading away from the root.

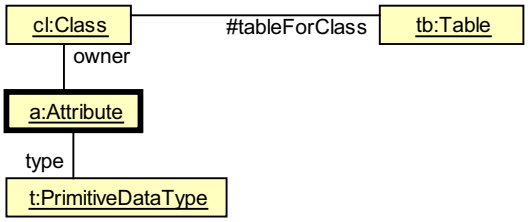


Figure 5. Associations in a pattern.

The simple patterns described so far do not require a more formal match definition. However, for extended patterns in section 5 such a definition will be used.

Here one principle of a good programming style in MOLA should be given. To achieve a high execution efficiency, pattern associations leading away from the loop variable (the pattern root) should have the **0..1 multiplicity** at this end in the corresponding metamodel. This means that we test the **existence of one possible instance**. In addition, in the case of existence, the match is unique then and we can reference this instance for various purposes, e.g., to use its attribute values in a deterministic way. Actually, all examples in the paper use this principle. For other multiplicities the extended patterns in section 5 serve well.

Patterns can use also the **reference** notation – an element whose name is prefixed by the @ character – this means that an already selected instance (by a previous statement, typically a loop head) must be used. This way patterns can be structured – similarly as, e.g., in [11]. See an example in Fig. 8.

4.2. Actions of the rule, complete rule examples

Pattern matching is only one part of the rule application. Another one is to perform the actions specified in the rule (on the basis of the current match). These actions modify the current instance set – typically, the target model. The following actions can be specified in a rule:

- building new class instances
- building new association instances (connecting new as well as existing instances)
- changing the attribute values – both for new and existing class instances
- deleting instances

The **action specification** (the "RHS part") of a statement has a structure similar to the pattern. It also consists of elements to build (in the instance notation) and

associations linking the new elements between themselves or to the pattern elements. Syntactically the action part is distinguishable by dotted lines and the line color – it is in **red**. Actions can also specify the deletion of an existing element (matched by the pattern) – this is shown by dashed lines.

The most typical action is the **building** of a **new class instance**. Building of class instance in MOLA is always accompanied by building of one special association – the **mapping association**, which in the rule must be linked to a pattern element (e.g., an association with the role name *#tableForClass* is linked to *cl:Class* in Fig.6). The role name of this association is specified in the intermediate metamodel (here – Fig.3) if that exists, but anyway its name must be prefixed by the # character. At the instance level it means that one instance of the new class is built and linked by the mapping association to the existing instance of the corresponding pattern element.

One goal of the mapping association is to serve for matching in the patterns of next rules. It is very typical in MDA model transformations that the transformation of a "higher level" element – package, class etc. determines how its subordinates – classes, attributes etc. must be transformed. The mapping association is namely the element linking such subordinate rules and ensuring their consistency. In addition, the mapping association reifies physically the mapping between the source and target model (and serves for tracing), hence such a name is used for this concept in MOLA. In our simple patterns, the cardinality of the mapping association is 1 – 1, but in more advanced patterns of MOLA it may have cardinality 1 – 1..* (and serve for determining the instance set of the new class which must be built).

The remaining action element is the **assignment** of **attribute values** (done by Pascal-like assignment statements). For an attribute to be set the new value is defined by an OCL style expression, which can contain one extension – attributes from pattern elements may be referenced, just by prefixing them with the instance name. The semantics is straightforward – take the attribute value from the existing instance matched to the element. The attribute assignments can be done for the "new" instances, but attributes of existing instances (in the pattern elements) can also be modified this way.

Fig.6 shows a complete example of a statement in MOLA. This is the first statement in the program for building the database definition from a class diagram.

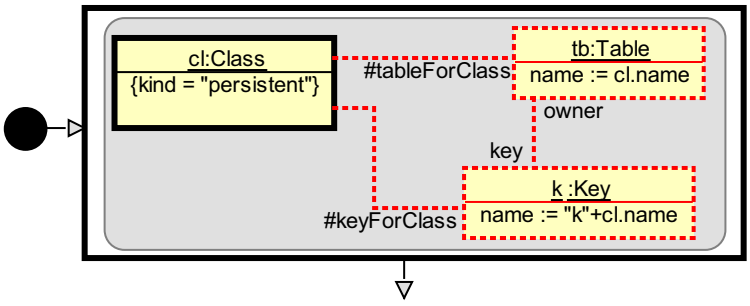


Figure 6. Simple statement in MOLA.

This statement is a FOREACH loop consisting of its loop head only, this loop head is also a rule which builds new instances. It does the first job in the transformation process – builds instances of both *Table* and *Key* for any *Class*

instance whose *kind* has the value *persistent*. The *name* attribute in each of the new instances is set to the specified value – to the *name* value in the matched *Class* instance. In addition, the two mapping associations are built, as well as an association instance with the roles *key – owner* between the new instances.

Fig. 7 shows the next two statements of the transformation program – both FOREACH loops too. The first one builds columns corresponding to primitive-typed attributes of persistent classes from the source model. Its pattern (discussed in section 4.1) selects the appropriate table in the target model (built by the previous statement) and the rule associates the new column to it.

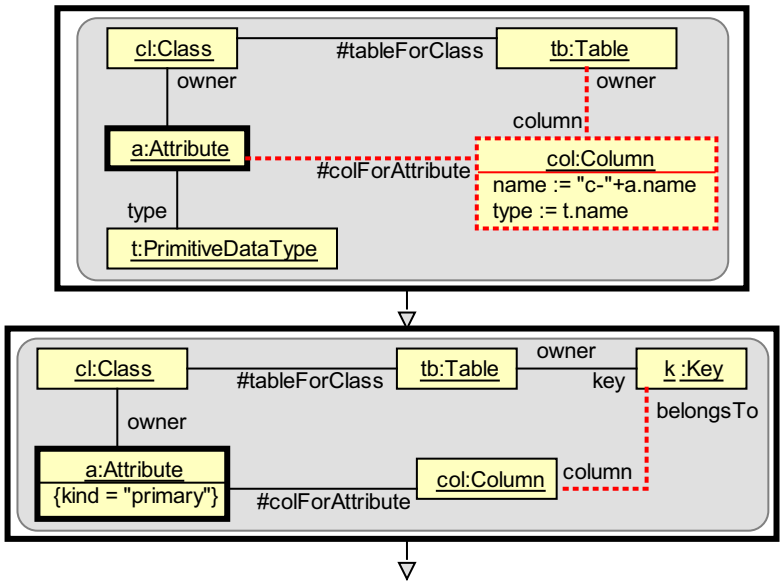


Figure 7. Statements transforming primitive attributes to columns and associating key columns.

The other statement in Fig.7 has the intention to attach the *belongsTo* association to each *Column* which corresponds to an *Attribute* with *kind = primary* (the other end of the association must be the *Key* for the relevant *Table*).

One more task to be done is to process associations in the source model. Fig.8 shows the corresponding program statement – a nested FOREACH loop. The top level loop builds the foreign key for each association in source model and associates it to the relevant primary key (built by the first statement). The nested loop builds a new column (in the table corresponding to the source class) corresponding to each column in the referred primary key. The pattern of this loop uses three references to instances selected in the top loop – all prefixed by the @ character. We remind that this means that namely these referenced instances must be used in any match for the subordinate pattern. Thus there is no need to repeat the corresponding selection conditions in the nested loop – its pattern becomes simple.

Certainly, the building of columns for a foreign key could be done by a separate independent loop, but then its pattern would be more complicated. In general, a certain "**breadth first programming style**" – each action a separate top level loop –

is in most cases usable for typical MDA jobs. But sometimes nested loops help to structure complicated patterns.

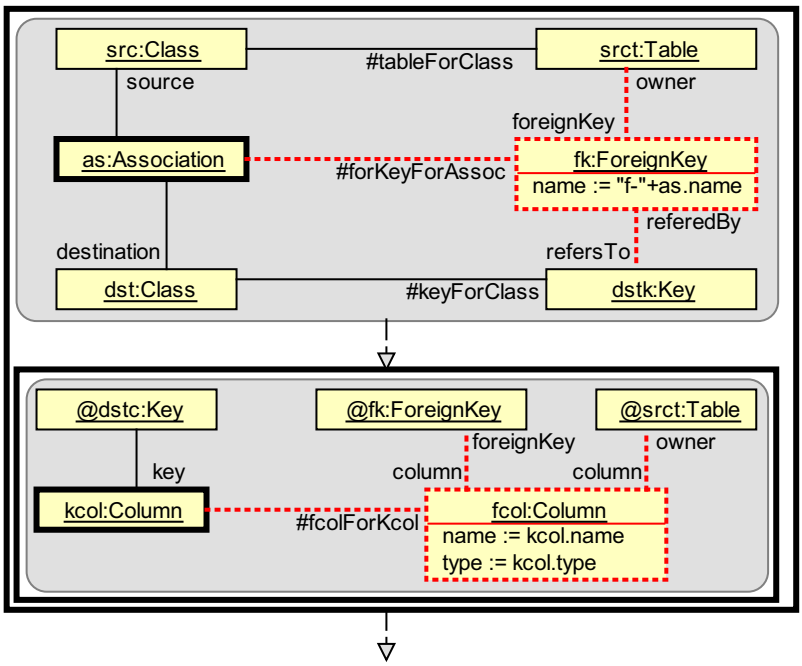


Fig. 8 Statement processing source model associations.

The remaining task – flattening class-typed attributes can also be implemented in the basic MOLA described so far. However, since the flattening is a true transitive closure task, and in its most complicated form – find all possible paths in the source model from a class to its indirect primitively-typed attributes and compute a name along each of the paths, it requires creating copies of attribute instances, building temporary associations and attributes and using depth three loops. In other words, the standard algorithm for building all paths in a graph has to be implemented. An alternative more readable solution is to use extended patterns to be considered in the next section.

5. Extended patterns

The patterns considered so far have one limiting property – only one instance can be matched to an element. Since this is too restrictive for some tasks in real transformations, especially those related to transitive closure, various ways to extend the pattern notation will be considered in this section. First, patterns with cardinality constraints will be considered – they may have unlimited number of instances associated with one pattern element, but the matching depth is still limited. An efficient match building procedure for this case is defined in a completely different way – as a stepwise algorithm on a graph. The efficiency of this procedure is

guaranteed if uniqueness principle is observed – pattern cardinalities match to the metamodel multiplicities. Finally, the looping patterns with unlimited matching depth are introduced – namely these patterns are more powerful than those in [10, 11] and permit to perform nontrivial actions, including transitive closure, in one rule.

5.1. Cardinality constraints for navigation associations

In section 4 the simplest case was considered where each pattern element was matched to a single instance and each association to a single association instance linking the matched class instances. In order to have larger fragments of the instance set mapped to the pattern, with several instances associated to one pattern element (what is really required by transformation rules), the extended MOLA uses **cardinality constraints** attached to pattern associations (actually something similar is used also in [10, 11]). In addition, the associations with cardinality constraints are treated as **directed** graph edges – using the UML navigability notation.

One of the constraints - **ALL**. It is used when the association has * or 1..* multiplicity at the appropriate end in the metamodel. It corresponds to * in UML – take all what you can, but nothing bad, if none. Another constraint is **OPT**. It is used with associations having 0..1 multiplicity at the appropriate end and means – take the instance if it exists, but the pattern does not fail if none exists. Actually OPT is a decorative version of ALL for the 0..1 case – to improve the readability. And we remind that empty cardinality constraint actually means **just one**. **NOT** also can be used as a cardinality constraint – there is none. In fact, there are constraints in MOLA which correspond to all possible UML multiplicities, but currently we don't need the other.

Now, more precisely, what is a valid extended MOLA pattern. Here we consider only the case when the pattern is used as a loop head. Then there is a loop variable, which serves as a pattern **root**. The pattern consists of **two** parts, having the **loop variable** as the **sole common node**. The first part, which uses undirected associations (and no additional cardinality constraints), is the same as before. It expresses (as before) the conditions for selecting valid instances for the loop variable. The other – the **extension** part uses **directed** associations and cardinality constraints and is used for matching to a set of instances. It must be a **directed acyclic graph (DAG)** starting from the loop variable as a root (a more complicated case with loops is considered in the next section). This part can be built by taking classes from the metamodel (in fact, IMM), converting them to pattern elements (adding instance names) and adding metamodel associations as directed edges. The extension part must be **distinguishable by associations** – if several edges with the same role name leave a node, they must lead to different classes (this requirement is essential for having an efficient match procedure, a weaker version of this restriction will be given in 5.2). The next step is to add appropriate cardinality constraints to the **navigable ends** of associations – ALL if the multiplicity in the metamodel is * or 1..* and OPT or nothing (one) if multiplicity is 0..1 or 1. When cardinality constraints are set this way, we say that the "**uniqueness principle"** is observed (the pattern fits to the metamodel). The pattern in Fig. 9 obviously satisfies the uniqueness principle – ALL is at the *attribute* end of the association from *Class*

(where the multiplicity in IMM is *), all other multiplicities are 0..1. To emphasize the fact that we expect many instances of *Attribute* to be matched, a decorative element in the pattern – the **multiobject** notation (from UML collaborations) can be used for the *Attribute* element.

We make here also one assumption – the extension part edges leaving the root node have the constraint ALL or OPT (the extension part should express an unlimited, but optional at the same time part of the match).

Let us consider an example for the usage of ALL constraint in an extended pattern – the first statement from Fig.7 but defined in an alternative way – in Fig.9.

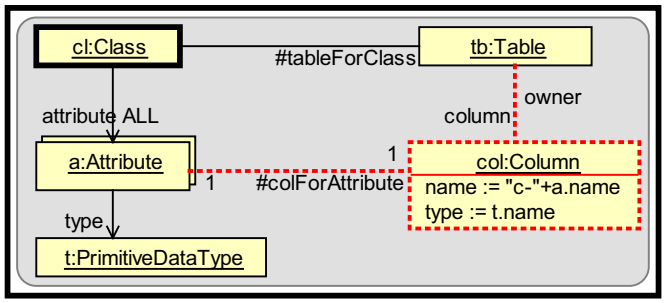


Figure 9. Rule with cardinality constraint.

The loop variable accepts as valid those instances of *Class*, for which a *Table* has been built. Now, when the loop is executed for a valid instance of the loop variable, the following new action is performed. For the extension part of the pattern (containing the elements *a:Attribute* and *t:PrimitiveDataType*) a temporary instance **DAG** is built containing all matches for the given root instance. For the given example this DAG is very simple – **all** those *Attribute* instances for the given *Class* instance (root), which are linked to a *PrimitiveDataType* instance, together with the corresponding association instances (*attribute* and *type*). The result is indeed a DAG, but not a tree, because a *PrimitiveDataType* instance can be used for several *Attribute* instances. Here the original instances from the source model are used as nodes for the DAG (we can assume that the nodes and edges of the DAG are highlighted, e.g., by "coloring" them green), but in some cases node copies are built – see section 5.2.

When the instance DAG is built, the rule actions are performed. Here a *Column* instance is built for each *Attribute* instance in the DAG (i.e., an instance associated to the element *a:Attribute*). This is specified by the mapping association (*#colForAttribute*) which now has an explicit multiplicity 1-1. Then the DAG is discarded (the highlighting is removed).

Now we will define the match building for an arbitrary pattern. The most natural way is to use a **procedural match definition.** We treat the pattern (its extension part) as a DAG from the root and build the instance DAG starting from its root – the current loop variable instance. Valid instance nodes are added to it layer by layer, in strict accordance with the pattern – so that each instance node can be assigned to a pattern node at that layer and the edges also match. Here the number of layers is fixed (determined by the pattern). Cardinality constraints must be taken into account – if the constraint is ALL, it doesn't matter how many edges exit a node in the

current layer, but for the default constraint (just one) there must be the corresponding edge to a node in the next layer. If that edge is not found, the node must be removed from the current layer as invalid. The pattern edges with ALL constraint actually generate fan-out cases in the instance DAG.

The formal definition of the **matching procedure** (generating a complete valid match – the instance DAG) is the following:

1. mark the current instance of the loop variable as the only instance in the layer one of the instance DAG and associate it to the pattern root.
2. take a node in the current layer of the instance DAG. For each directed association leaving the pattern node, to which the instance node is associated, find **all** association instances from the current node and select those where the target instance satisfies the corresponding attribute constraint; add this "filtered neighborhood" to the next layer. Repeat this for all nodes in the layer. If a node in the next layer has been reached twice, mark it only once (the path history is not important in this mode).
3. assign instances in the next layer to the corresponding elements in the pattern (it can be done uniquely due to the required distinguishability by associations).
4. check cardinalities – for each node in the current layer and for each navigation association (which has the default cardinality constraint - i.e., "just 1") from the associated pattern node check whether there is an instance of this association. If there is none, remove the instance node from the current layer, and recheck the previous layers (for layer one it cannot occur due to our assumption). ALL and OPT constraints require no check.
5. repeat steps 2, 3, 4 for each layer of the pattern

The semantics of ALL guarantees that always the maximal match is selected – no subset of a match can be a valid match. Even more, for a given root the procedure result is **deterministic** – it is due to the "uniqueness principle" for the pattern, that from several possible instances all are selected, and one instance must be selected from possible one. Namely this would permit also an efficient match implementation in MOLA.

5.2. Looping patterns

In this section we introduce the final elements of extended patterns in MOLA. First, we permit patterns to have **directed loops** in the extension part when a pattern is built on the basis of a metamodel fragment. This extension is essential for defining a **transitive closure in a pattern**. Features will also be provided for defining a closure involving all possible paths.

The requirement introduced in 5.1 that the extension part must be **distinguishable by associations** is still in place. This requirement is sufficient for the example in Fig. 10 and many similar ones. A weaker restriction – the K-distinguishability – sufficient for any reasonable MDA task will be considered at the end of section (however, it makes the matching procedure more complicated).

Certainly, the **uniqueness principle** from 5.1 must be observed when assigning cardinality constraints to pattern edges – violating this principle would lead to a much more complicated and inefficient matching procedure.

Though a pattern now may have loops, the instance graph for it will be required to be a DAG anyway. From the theoretical point of view, we may be interested in finding instance-level loops via patterns, but no MDA related job was found where it makes sense. Therefore loops at the instance level will be simply forbidden by the matching procedure.

One more remark refers to nodes of the instance DAG. In 5.1 a simple case was considered where the original model nodes were used (just highlighted). But this implies that the path history cannot be stored in the instance DAG – if a node is reachable via two or more paths, data from the path cannot be stored in the node. The only way for storing this data is to make copies of the original instances and store them in the DAG. In an extended MOLA statement it can be specified which **pattern nodes must be copied** (such a node is marked by a **square icon**) during the building of the DAG (it makes sense only for the looping nodes). The temporary copies in the DAG are related to their originals and "inherit" all attributes and association instances from them. When the statement completes, the temporary copies are discarded. Copying selected pattern nodes is the easiest way to implement transitive closures where all paths from a node must be traversed – as the one in Fig. 10.

The same **matching procedure** from the previous section is usable, but with the following extensions:

- step 2. New instance which is already present in the DAG on the path (from the root to the current instance) is not added to the next layer – a safeguard against infinite loops. If the target instance corresponds to a pattern element in the "copy list", make a copy of the instance and of the relevant associations, relate the copy to the original.
- step 5. Repeat steps 2, 3, 4 until no instances are placed in the next layer (the repetition is no longer limited by the number of layers in the pattern due to possible loops)

A typical use of a looping pattern is for performing a **transitive closure** along a metamodel association. Transitive closure is directly supported in the textual languages [12, 14], but no other graphical pattern languages [7, 10, 11] support it.

Fig.10 shows an example of a looping pattern. It is the last statement in the class to database transformation program and performs a recursive "flattening" of the class diagram – adding indirect attributes (columns) to a class (table), whose direct attributes have another class as a type. In this example actually a transitive closure on the *attribute* association is performed. The only loop in the pattern is formed by the *type* association from *a2*. The *type* edge can lead either to *t* or to *c2*, the situation is distinguishable because they are of different classes. Both of the edges have the OPT cardinality constraint. During the pattern match, just one of these possible continuations will occur (because an *Attribute* always must have a *type*). If no *type* leads back to a *c2*, then looping along this path is finished. If all the looping is finished, a large instance tree (it is indeed a tree due to instance copying, except the "terminal" instances of *t*) with the root *Class* as a root and certain number of *Attribute* instances as leaves is built as the result of the match. The tree represents all possible paths via indirect attributes from the given class instance - due to the copying of *c2* and *a2* two paths never join. The leaves have a primitive type – they will be used for columns. However, it should be noted that both leave and non-leave

Attribute instances are assigned to *a2* – they must be sorted out to find leaves only. This tree represents graphically the transitive closure of the *attribute* association.

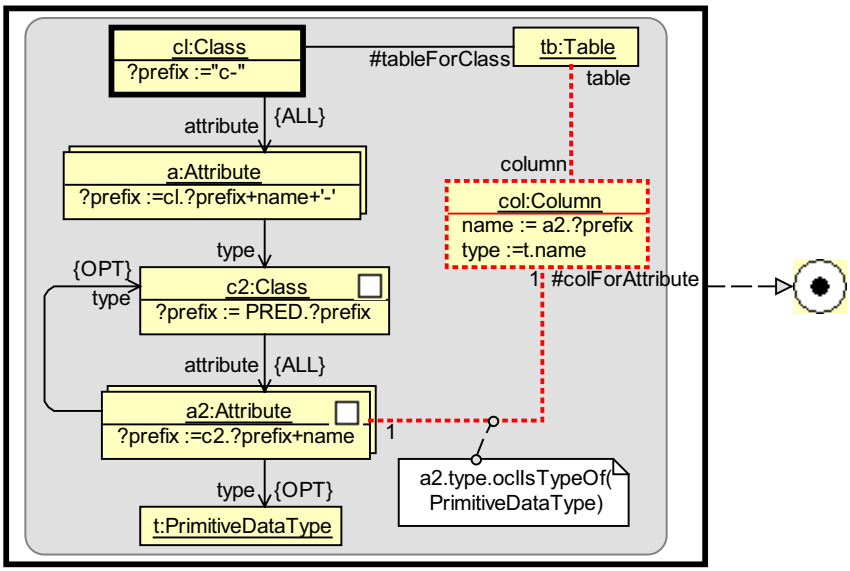


Figure 10. Rule with looping pattern.

Looping patterns typically involve complicated **assignments** to **computed attributes** of metamodel classes.

There are several kinds of computed attributes in MOLA, differing in their scope. Here the most used attributes are **statement-local attributes**, their scope being actions within one statement execution (here – one iteration of the loop, during which the extended match is built). Their values are discarded after the loop iteration is complete. Their main use is for finding various qualified or compound names, typically appearing in MDA tasks.

If the intermediate metamodel is used (here – Fig. 3), rule-local attributes are defined in it for all relevant classes, their **names start with** "**?**". Their computation is performed immediately after the pattern match – when the instance DAG is complete. Rule-local attributes are computed in the same order as the match itself was built – starting from the root and moving away from it. If the instance DAG contains copies, these attributes are located in copies – not in the original instances. The assignment statement for a computable attribute in its expression part can contain the value of this attribute in the predecessor node and any values of normal (source) attributes in the current node (these are unqualified). The attribute value from the predecessor node can be qualified either by its instance name or by a **PRED** keyword. The use of PRED is required in cases when a node may have several nodes as predecessors – due to loops in patterns. Each node in the path must have the corresponding assignment statement, otherwise the attribute computation is terminated there. Several attributes may be computed simultaneously in a rule.

The example in Fig.10 contains assignments for the single computed attribute – *?prefix*, which is contained in *Attribute* and *Class*. The computation starts in the

root, where the constant value – the string "*c*-" is assigned. When the value is propagated through an *Attribute*, the value of its *name* and the constant "-" is concatenated to it. The propagation through the *Class* node does not change the value. It is easy to see, that in the result the value of *?prefix* for each leaf of the match tree is the concatenation of all *Attribute* names along that path, separated by "-" and prefixed by "*c*-" – namely the value specified in this task as a *Column* name for indirect attributes. The obtained values are used namely for this purpose – they are used in the assignment for the *name* of the new *Column* instance. A *Column* is built only for those instances assigned to *a2* which are of primitive type (are leaves in the tree) – this is specified by the OCL constraint at the mapping association.

Fig. 10 completes the example transformation program in extended MOLA – the complete transformation consists of Fig. 6, 7, 8 and 10.

We conclude the section with a weaker restriction for patterns, than the distinguishability by associations. Namely, if in a pattern an association with the same role name can lead to several nodes, these nodes must be **K-distinguishable** – their neighborhood of order <=K (in the sense of directed graphs) must contain **mutually exclusive** elements – different local constraints, different mandatory (just one) associations, mandatory associations leading to different classes etc. The distinguishability by associations can be considered to be 0-distinguishability.

For most MDA examples – e.g., more complicated versions of the example in this paper, and many similar ones, typically K is 1 or 2. In order to use patterns with K-distinguishable elements, a **look-ahead** (in the instance set, but along the pattern edges) not longer than K has to be included in the matching step 3.

6. Conclusions

The paper describes the basic principles of the graphical model transformation language MOLA. There are two innovative elements in MOLA. One is natural combination of simple control structures with pattern based rules. The other one is the powerful pattern mechanism supporting variable cardinality and looping patterns, thus enabling transitive closure in patterns and simplifying even more the control structure of the language. The complete language MOLA is tested on several real world MDA examples, such as converting statecharts to FSM and realistic class-to-database transformation including class inheritance, transformation of business process models to workflows etc. The results show that in most cases more compact and readable rule definitions have been obtained, when compared to e.g., pure recursive style in [10]. The extended patterns permit to strike a right balance between complexity of rules and control structures governing them, thus providing the required transformation readability. Simple recursions, typical e.g., to statechart flattening job, can be specified in a readable way using the WHILE loop in basic MOLA.

The extended patterns, though more complicated than patterns of basic MOLA, are defined in a way to permit also an efficient implementation.

Acknowledgements

Authors of the paper are grateful for valuable discussions and comments provided by their colleagues at the IMCS Modeling and Model Transformations seminar. This research was partially funded by Science Council of Latvia under the project Nr.02 0002.

References

[1] OMG: MDA Guide Version 1.0.1, http://www.omg.org/docs/omg/03-06-01.pdf
[2] OMG: Unified Modeling Language: Superstructure. Version 2.0 (Final Adopted Specification). http://www.omg.org/cgi-bin/doc?ptc/2003-08-02 (2003)
[3] Booch G., Jackobson I., Rumbaugh J. The Unified Modeling Language. Reference Manual, Addison-Wesley, 1999.
[4] OMG: Meta Object Facility (MOF) Specification. Version 1.4. http://www.omg.org/cgi-bin/doc?formal/2002-04-03
[5] OMG: Request For Proposal: MOF 2.0/QVT. OMG Document ad/2002-04-10, http://www.omg.org/cgi-bin/doc?ad/2002-04-10
[6] Bettin J. Ideas for a Concrete Visual Syntax for Model-to-Model Transformations. Proceedings of the 18th International Conference, OOPSLA'2003, Workshop on Generative Techniques in the context of Model Driven Architecture, Anaheim, California, USA, October 2003.
[7] QVT-Merge Group. MOF 2.0 Query/Views/Transformations RFP, Revised submission, version 1.0. OMG Document ad/2004-04-01, http://www.omg.org/cgi-bin/doc?ad/2004-04-01
[8] Compuware, SUN. MOF 2.0 Query/Views/Transformations RFP, Revised Submission. OMG Document ad/2003-08-07, http://www.omg.org/cgi-bin/doc?ad/2003-08-07
[9] Interactive Objects Software GmbH, Project Technology, Inc. MOF 2.0 Query/Views/Transformations RFP, Revised Submission. OMG Document ad/2003-08-11, http://www.omg.org/cgi-bin/doc?ad/2003-08-11
[10] Willink E., A concrete UML-based graphical transformation syntax - The UML to RDBMS example in UMLX. Workshop on Metamodelling for MDA, University of York, England, 24-25 November 2003.
[11] Agrawal A., Karsai G, Shi F. Graph Transformations on Domain-Specific Models. Technical report, Institute for Software Integrated Systems, Vanderbilt University, ISIS-03-403, November 2003.
[12] DSTC/IBM/CBOP. MOF Query/Views/Transformations RFP, Second Revised Submission. OMG Document ad/2004-01-06, http://www.omg.org/cgi-bin/doc?ad/2004-01-06
[13] Celms E., Kalnins A., Lace L. Diagram definition facilities based on metamodel mappings. Proceedings of the 18th International Conference, OOPSLA'2003, Workshop on Domain-Specific Modeling, Anaheim, California, USA, October 2003, pp. 23-32.
[14] Kleppe A., Warmer J., Bast W. MDA Explained. The model driven architecture: practice and promise. Addison-Wesley, 2003.

SCIENTIFIC PAPERS UNIVERSITY OF LATVIA. 2004. VOLUME 672, Datbases and Information Systems
357–372 P.

Design Independent Modeling of Information Systems Using UML and OCL

Lina Ceponiene, Lina Nemuraite

Kaunas University of Technology, Faculty of Informatics,
Department of Information Systems
Studentu 50-308, Kaunas, Lithuania
{kavalina, nemur}@soften.ktu.lt

Abstract. Design Independent Model (DIM) is proposed for definition of requirements for development of wide range of information systems conceived as processes or systems composed of services. It is defined in terms of actors, interfaces to target system (for development of e-services) or interfaces to process (for development of e-business processes), models of problem domains and sub-domains, related with interfaces. Metamodel of Design Independent Model presents subset of UML 2.0 metamodel with additional constraints. The kernel for ensuring relative completeness and compatibility of requirements is schema, in which interactions and state transitions related aspects of behavior are integrated with static structures of entities of problem domain. Direct and reverse procedures for mapping between sequence diagrams and state charts are presented that help reveal inconsistencies and incompleteness in definition of required behavior. Possibilities for implementation in CASE tools are investigated, case study and illustrating examples are presented.

Key words. Information systems, metamodel, UML, OCL, interface, state machine, interaction.

1. Introduction

Information systems (IS) development is diverging from object-oriented to model-driven, where models and metamodels are playing central role instead of objects [9]. Also, the shift to separation of behavioral objects (services and processes) from informational objects (entities) is observed as opposed to object-oriented principles of tight binding structural and behavioral features together. This shift is raised by mismatch between business and software (separation of processes from entities), or between practical software paradigms and object-orientation (e.g., sending and retrieving messages from the queue instead of invoking operations on objects).

Today, two different processes in development of IS may be accentuated: creation of IS services (e-services or, more specifically, Web services [28]) and creation of e-business processes for management of e-services. Separation of process management from business functions carried out by different participants of process was the principle idea of Workflow Management Systems (WFMS). Today

this idea has penetrated in Enterprise Application Integration (EAI), WFMS, Business Process Modeling and Execution languages (BPML, EBXML, BPEL, etc.) as general methodology for software development and exploitation.

Another great methodological shift is Model Driven Architecture (MDA) [9], [22]. The essence of this approach is in idea that the foundation of software development is Platform-Independent Model (PIM) of software system. PIM is obtained from Business Model (formerly Computation Independent Model, or CIM) and serves for definitions of multiple Platform Specific Models (PSM), from whose implementations on different platforms may be generated. This idea equally applies to development of IS services as well as of e-Business Processes.

In this paper Design Independent Model (DIM) is proposed for definition of requirements for development of wide range of information systems conceived as processes or systems composed of services. In our opinion, there is a gap between Business Model and Platform Independent Model. PIM, platform independent model, is dependent upon software architecture and design methodology. In contrast, DIM represents comprehensive definition of requirements – structure and behavior of IS independently of software architecture (e.g. object-oriented, component-based, service-oriented, Web) and design method (e.g. data-driven, event-driven, responsibility-driven etc.). Independency from design implies that operations are not allocated to control classes or components but defined rather as events triggering interface objects. The eventual purpose of DIM is formalization of process going from requirements to PIM, but it goes beyond limits of this paper. Introduction of DIM leads to transparency of design decisions that must be made during development of PIM.

DIM is defined in terms of actors, interfaces to target system (for development of e-services) or interfaces to process (for development of e-business processes), models of problem domains and sub-domains, related with interfaces. In general case it may be represented visually by stereotyped UML [11][25] class, sequence, state charts and activity diagrams with OCL constraints [27] [12] [7] [26] following principles of precise modeling [23], [16] and contract-based development [8]. These diagrams are partially overlapping views under DIM that actually defines schema of IS [10], fully representing state and behavior of IS under development in design-independent manner [5].

In current paper DIM is limited to creation of IS services and, respectively, usage of class, sequence and statechart diagrams. For development of stateless services there are no requirements for explicit management of process (in some cases, we may deal with services that are not joined by common process at all). In such case activity diagrams are not mandatory, though in being of common process they have explanatory importance and may be used in beginning of DIM development or generated from DIM when it is completed. In opposition, activity diagrams play the crucial role in development of IS where processes must be managed as they are able to integrate data flow and control aspects of behavior together with actors roles participating in process.

The particular attention is devoted to integration of interactions with state machines, which are specialized as state machines of entities and interfaces to behavior. Explicit separation and reconciliation of states of entities with states of services is similar to [24]. Interactions are not directly integrated with state machines in UML metamodel. Such integration is necessary for harmonization of IS

model irrespective of development phase (requirements, design or implementation). In this work algorithm is proposed for direct and reverse transformations between sequence diagrams and state machines. There are many works on this topic (e.g., [4], [14], [29]), but bi-directional transformations were not possible. For this purpose some adjustments to UML metamodel must be made.

Models and algorithms presented in this paper, if implemented in UML CASE tools, would have effect on design practice, and not necessarily for development in MDA. Interactive generation of lacking diagrams from initial ones, caution about deficiencies would help designer to develop compact, unambiguous models and keep them in line instead of rapid sketching multiple uncoordinated views.

The paper is composed of 5 sections. In section 2, the structure of Design Independent Model of IS in general case is defined and metamodel integrating class, sequence diagrams and state charts is presented. Section 3 is devoted for derivation of state machines from sequence diagrams and vice versa. Section 4 introduces the case study illustrating the usage of DIM. Finally, section 5 concludes the paper and discusses possible future work.

2. Design independent model

Commonly use cases are used to capture behavioral requirements of software systems, but they often are textual descriptions in natural language and depend on individual interpretation, so are ambiguous [13] [20]. In this work model of use cases and their textual descriptions (initial requirements) are the input for creation of elaborated requirements model, i.e., DIM.

Model for comprehensive definition of requirements proposed in this work is based on many sources (e.g. [1], [3], [4], [19] etc.]. At conceptual level it was introduced in [5]. Elaborated model of requirements must define state and behavior of IS independently of future design. Behavior of IS has many aspects: interactions, state transitions, control flows and data flows. Different kinds of UML diagrams are devoted for representation of these different, often overlapping aspects, but all these diagrams are views under the same model of target IS. In requirements phase this model, named DIM, is specification, containing essential elements of IS model, which may be visually represented using class, sequence, state charts and activity diagrams (Fig.1).

In DIM, elaborated use cases are specified as interfaces to the target system together with actors – service users and external systems (service providers), whose interfaces must be accessed by the target system to provide requested services. Such form well conforms to requirements for development of components and Web services but it also may be applied for any IS or software system. Further, interfaces are specified by the set of operations identified from steps in use case specifications. Every operation is defined by its signature, pre and post conditions (e.g., [8], [6] etc.). Similar approach was proposed in [21], where use cases are supplemented with operational schemas, but grouping of operations is lost. In this work, relationships between use cases (<<include>>, <<extend>> and generalization) are presented as generalization relationships between interfaces strongly following Liskov substitution principle [15]. Other possible relationships between interfaces are

dependencies with stereotype <<use>>. Usage dependencies [25] describe relationships between service users and interfaces, interfaces and external service providers, and different interfaces. The last kind of dependency may be discovered during use case analysis when sequence diagrams are created for modeling interactions between actors and interfaces.

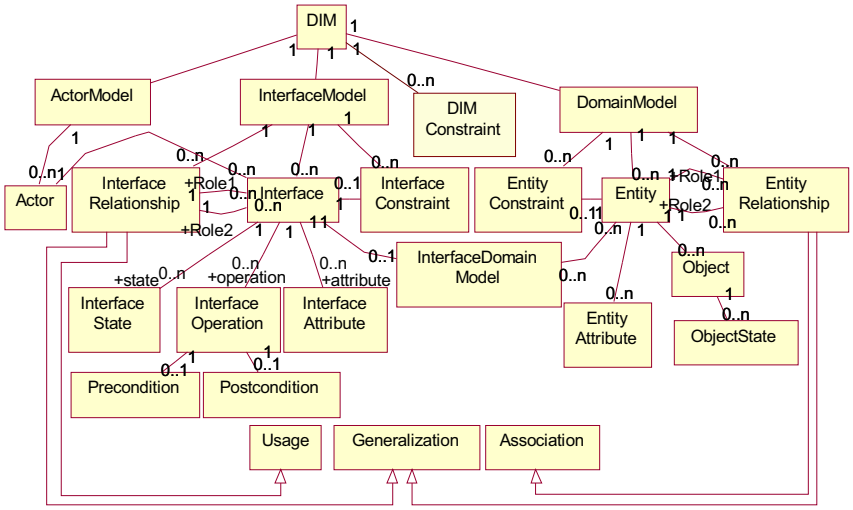


Figure 1. Metamodel of Design Independent Model

Besides the interfaces, DIM includes entities and states of entities modeling the state of problem domain and sub-domains associated with particular interfaces. Possible relationships between entities are associations and generalizations, as proposed by authors of executable UML [23]. Entity and interface constraints (invariants) also are included in DIM.

2.1. Sequence diagram for representation of requirements

For development of DIM, one or more sequence diagrams for every use case are constructed from specifications of use case scenarios. These sequence diagrams must represent all desirable interactions between actors (service users), interfaces (i.e. use cases) of the system and possibly the interfaces to external systems, if they are needed to fulfill service requests. DIM deals with elaborated requirements; so initial use cases (or later interfaces) must be re-structured for reuse of behavior where advisable.

The main elements of the sequence diagram are *Lifelines* (used in UML 2.0 instead of classifier roles from previous versions of UML), *Messages* and *State Invariants*, representing states of the classifiers. Two types of *Messages* are distinguished: *Requests* and *Responses*, and two *Events* are connected to each message in sequence diagram: *Send Event* corresponds to one message end and *Receive Event* corresponds to the other one (Figure 2). *Received Request* may have attached *Constraint* that must be verified before fulfillment of *Request*.

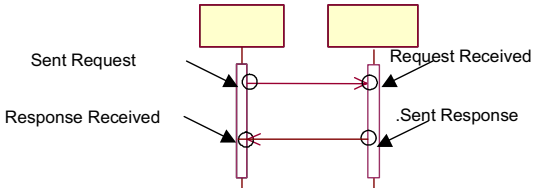


Figure 2. Event types

Sequence diagrams in DIM are based on UML 2.0 but have some additional adjustments. Metamodel of sequence diagrams, used for DIM, is presented in Figure 3 (there *Event* corresponds to *EventOccurence* in [25]). Messages are accepted to be synchronous as well asynchronous. For synchronous communication analysts often suppress *Response* messages in diagrams since *Response* implicitly follows after *Request*. For unambiguous representation of interactions *Response* messages are important and for synchronous messages must be included by default. Association connecting *Request* and *Response Messages* is added to UML metamodel, thus ensuring the possibility of tracking which *Response* is the answer to which *Request*.

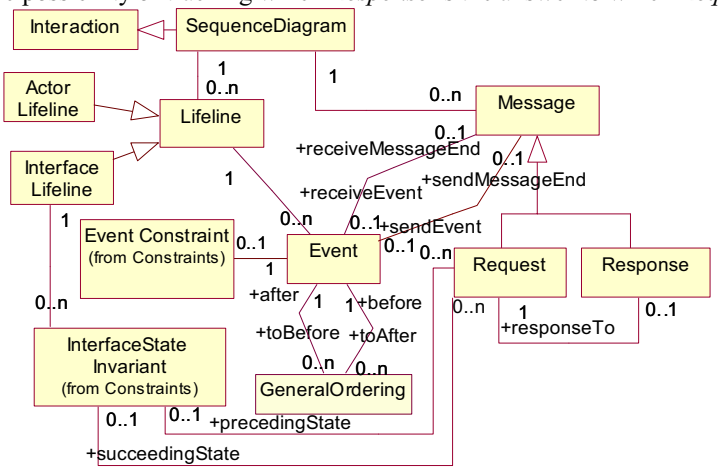


Figure 3. Sequence diagram metamodel for DIM

Request Messages received on *Lifelines* are associated with state invariants of this *Lifeline* by associations *precedingState* and *succedingState,* because *Received Request* messages correspond to the transitions between states of interface. *State Invariants* describing these states are associated with *Received* (preceding) and *Sent* (succeeding) *Request Messages*, and succeeding *Request Message* is associated with all following sequence of messages including *Response* to it.

Constraints for DIM sequence diagram metamodel:

– If there is a *Message* in the sequence diagram there must also be at least one *Lifeline* in this diagram too:

```
context SequenceDiagram
inv:self.Message->notEmpty() implies self.Lifeline->notEmpty()
```

– *Message* sender and receiver *Lifelines* belong to the same sequence diagram:

```
Context SequenceDiagram
```

```
inv:
self.Message->forAll(m|self.Lifeline->
exists(ls| ls = m.sendEvent.Lifeline) and self.Lifeline->
exists(lr| lr = m.receiveEvent.Lifeline)
```

– Every *Event* may be associated only with one *Message*, received or sent:

```
Context Event
inv:
self.sendEvent -> notEmpty() implies self.receiveEvent→isEmpty() or
self.receiveEvent -> notEmpty() implies self.sendEvent→isEmpty()
```

– Every synchronous *Request Message* must have *Response*:

```
Context Message
inv:
(self.MessageSort = synchCall or self.MessageSort = synchSignal)
and self.oclIsKindOf (Request) implies self.Response → notEmpty()
```

2.2. State machines for representation of requirements

State charts define the dynamic behavior of objects of the corresponding classes during their entire lifetime. Protocol state machines are proposed for describing the behavior of an interface [25], but they do not describe events sent to other state machines. Capturing of sent events is important for definition of interactions with external systems. This possibility is included in state machines in DIM (Figure 4).

Relationships between *Event* class and its sender and receiver must be captured in state machines; otherwise it would be impossible to trace interactions from statechart diagrams. These relationships are established as roles of classifiers that correspond to lifelines on sequence diagram.

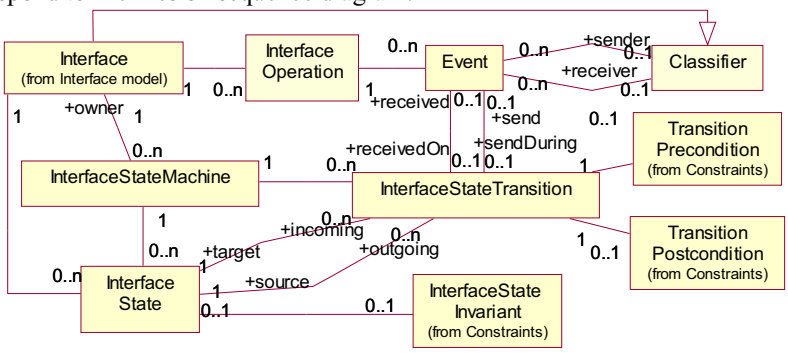


Figure 4. Interface state machine metamodel

Constraints of DIM state charts metamodel:

– If there is a transition in state machine there also must be at least one state in this state machine too:

```
context InterfaceStateMachine
inv:
self.InterfaceStateTransition -> notEmpty() implies
self.InterfaceState -> notEmpty()
```

– Event must have only one association with classifier – *Sender* or *Receiver*:

```
Context Event
inv:
self.sender->notEmpty() implies self.receiver->empty()
```

```
self.receiver->notEmpty() implies self.sender->empty()
```

– If transition has received *Event* associated with *Request Message*, it cannot have send event associated with *Response Message*:

```
Context InterfaceStateTransition
inv:
self.received.receiveMessageEnd.oclIsKindOf(Request) and
self.send.sendMessageEnd → notEmpty() implies
self.send.sendMessageEnd.oclIsKindOf(Request)
```

2.3. Relationships among sequence, class and statechart diagrams

Elements of class, sequence diagrams and state charts are interlinked in UML metamodel, but relationships among them have a high degree of abstraction. Such flexibility supports different kinds of usage of UML models, but in our case it is necessary to tie models for assurance of consistency between different views of the same system.

Fragment of metamodel, integrating state charts, class and sequence diagrams, is presented in Figure 5. Here state charts are integrated with sequence and class diagrams via events triggering operations of interfaces from interface class diagram. Interface class diagram is integrated with domain model of the system as arguments of operations of the interface represent entities, attributes and data types from domain model (otherwise they are common built-in types). Operation constraints are written using states of entities from problem domain. Transitions of states of interfaces and entities are integrated via events associated with states and transitions. Transitions of states of entities occur in states of interfaces; states of entities constrain transitions of states of interfaces.

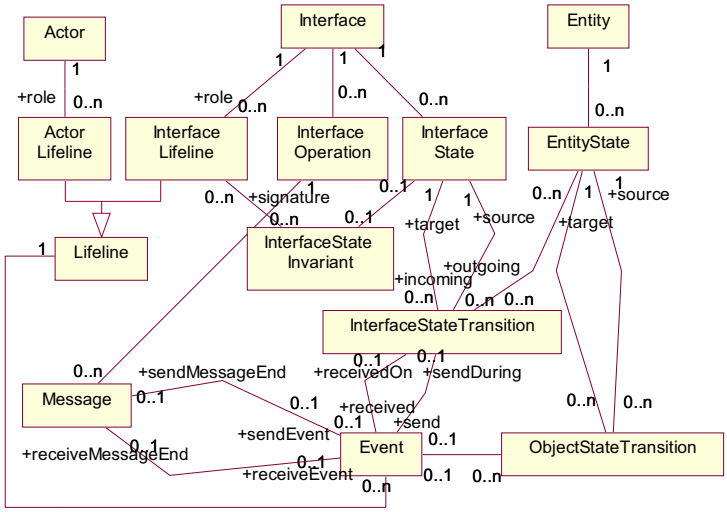


Figure 5. Fragment of metamodel integrating state charts, class and sequence diagrams

Part of constraints for DIM is represented on Figure 6. Using of these constraints removes ambiguity from relationships between representations of interactions and states. Similar constraints must be hold between sequence diagrams and class

```
   Context DIM
   Inv:
   self.InterfaceModel.Interface →
-- For every interface default state (e.g.'WaitState') exists
   forAll(i|i.InterfaceState → exists(s0|s0.name = 'WaitState') and
-- For all Lifelines of Interface
   forAll(i|i.InterfaceLifeline → (l|
   forAll(e|l.Event→  --For all Lifeline Events
-- If Event is Received Request, Event State exists
    if e.receiveMessageEnd.oclIsKindOf(Request)then
     i.InterfaceState → exists(s|s.name = e.name.concat ('State') and
--  If Received Request Event is first event on Lifeline or it has
--preceding Sent Response, Transition exists from Wait State to event
-- State and Event is Received on this Transition (Fig.7)
     if e.toBefore.before → isEmpty() or e.toBefore.before →
     exists (e1|e1.sendMessageEnd.isKindOf(Response))
     then s.incoming →
        exists (t|t.source = s0 and t.target = s and t.received = e)
     endif and
--  If Sent Request succeeds Received Request Event, then Request is
-- sent during Transition to Event State (Fig.8)
     if e.toAfter.after.sendMessageEnd.isKindOf(Request)
     then t.send = e.toAfter.after
--  If Sent Response succeeds Received Request Event then transition
-- exists from Event State to Wait State
     else if e.toAfter.after.sendMessageEnd.isKindOf(Response)
       then s.outgoing →
        exists (t1|t1.received = e.toAfter.after and t1.source = s
        and t1.target.s0)
       endif
     endif
--  If Received Request is asynchronous type then transition exists
-- from Event State to Wait State (Fig.9)
    if (e.receivedMessageEnd.MessageSort = asynchCall or
        e.receivedMessageEnd.MessageSort = asynchSignal))
      then s.outgoing → exists (t1|t1.source = s and t1.target = s0)
    endIf )
   endIf
--  If Event e is Received Response then State exists having incoming
--  Transition during which Request for this Response was sent. This
--  State is the state of the last Received Request event before
-- Received Response event e (Fig.10)
  if e.receiveMessageEnd.IsKindOf(Response)
     then i.interfaceState → exists
      (s|s.name = e.toBefore →
    select(e1|e1.receiveMessageEnd.isKindOf(Request))
                    → last().name.concat('State') and
-- If Received Response is last Event on Lifeline or it has succeeding sent
-- Response then Transition exists from this State to Wait State and
-- Received Response is received on this Transition and its succeeding sent
-- Response is sent (Fig.11)
   if e.toAfter.after → isEmpty() or
         e.toAfter.after.sendMessageEnd.isKindOf(Response)
   then s.outgoing → exists (t|t.source = s and
       t.target = s0 and t.received = e and
       if e.toAfter.after → notEmpty()
        then t.send=e.toAfter.After endif)
--  If Received Response has succeeding Sent Request then recursive
--  Transition exists triggered by Response and Request is sent
-- during this Transition (Fig.10)
      else if e.toAfter.after.sendMessageEnd.isKindOf(Request))
    then s.outgoing → exists (t|t.source = s and t.target = s
       and t.received = e and t.send = e.toAfter.after)
       endIf
     endIf ) endIf endIf ))))
```

Figure 6. DIM constraints associated with sequence and state chart diagrams

diagrams, between state machines and classes; states of interfaces must be related with states of entities etc. Different kinds of constraints must conform too.

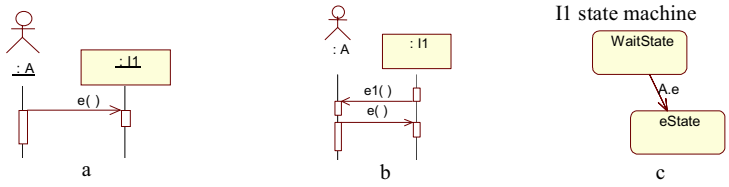


Figure 7. Representation of *Received Request Event*, first on *Lifeline* (a) or having preceding *Sent Response* (b), on sequence diagram (a, b) and on state machine (c)

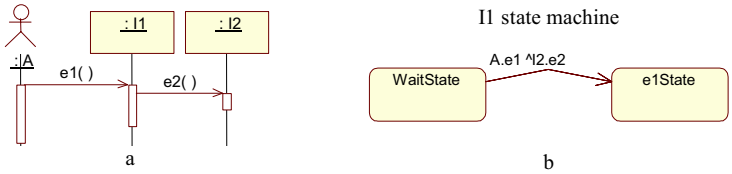


Figure 8. Representation of *Received Request Event*, having succeeding *Sent Request*, on sequence diagram (a) and on state machine (b)

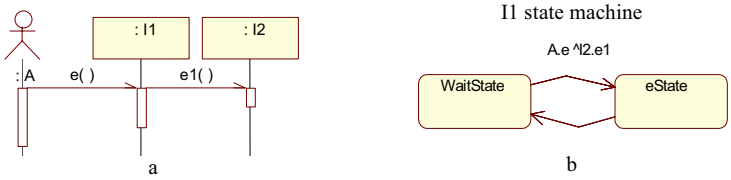


Figure 9. Representation of asynchronous *Received Request Event* on sequence diagram (a) and on state machine (b)

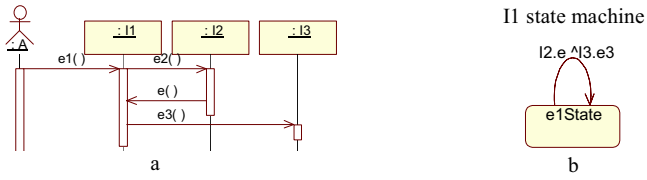


Figure 10. Representation of *Received Response*, having succeeding *Send Request Event*, on sequence diagram (a) and on state machine (b)

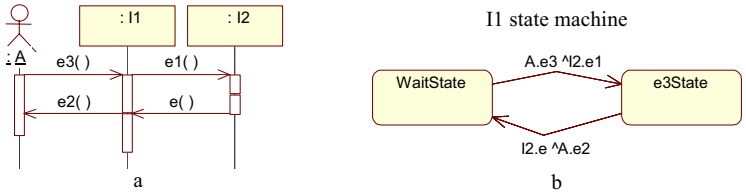


Figure 11. Representation of *Received Response Event* on sequence diagram (a) and on state machine (b)

In UML, it is possible to attach constraint to every element of model. In DIM, *Interface State Invariants*, *Event Constraints, TransitionPreconditions*, *Operation Preconditions* and *Post Conditions* are used. They have UML 2.0 semantics except *Event Constraints* that were used instead of *Interaction Constraints* [25]. Rules for consequent derivation of *Interface State Transitions Preconditions* and *Post Conditions* from *Event Constraints*, as soon as derivation of *Preconditions* and *Post Conditions* of *Operations* are elaborated, but they are not presented here because of limits of the paper.

Of course, pursuing of strict requirements loads with extra analytic work. CASE tool may help designer to keep different diagrams reconciled and to produce new diagrams from existing ones. Why is it needed? Because it is a way to see all peculiarities of the problem under investigation. If state machines would be derived from sequence diagrams, inconsistencies between interactions would be seen, as interactions of one classifier would be integrated from several sequence diagrams.

It is possible to proceed in reverse order – to make class diagrams or state charts and outspread them to interactions, then wrong traces would be clarified. More detail side of this kind of analysis is presented in the next section.

3. Reconciliation of Sequence and State Diagrams

The algorithm for derivation of state charts from sequence diagrams was developed that can be used for interactive generation of state chart designs from sequence diagrams or synchronization between state charts and sequence diagrams. The first version of this algorithm was presented in [17] and implemented using ArgoUML CASE tool. Developer, who manually creates state charts from sequence diagrams, can also use this process. Following the respective guidelines even in a case of non-automated derivation helps to maintain the consistency of the system description. The reverse process is proposed for obtaining sequence diagrams from state charts. It would help to test feasible scenarios of behavior defined by state machine, and to reveal missing ones.

3.1. From sequence diagrams to state charts

The derivation of state machine from sequence diagrams consists of two stages: obtaining the event sequences from the lifelines from sequence diagrams and synthesizing the state charts from these sequences.

The sequence (starting from the top of lifeline) consists of pairs where the first member corresponds to the state and the second member corresponds to the transition in state chart according to constraints (Fig.6). For example, every *Received Request* e_i corresponds to transition in state chart diagram; if *Sent Request* succeeds e_i it is connected to e_i as a *Sent Event* of the *Transition*, etc.

Event constraints are added to corresponding transitions. Information about the senders and receivers of events is captured in state machine, thus ensuring possibility to re-create sequence diagrams from state machines (this is not employed in UML state machines, although provided in UML metamodel). Sender may be described similar to receiver: *Sender.Event.*

The sequence is created starting with *oclVoid* as the first element of the first pair (it corresponds to the *Wait State* in the state machine) then goes the first *Received Request* e_i as the second element of a pair. If there is succeeding *Sent Request* it is connected to e_i as a *Send Event*.

For the second pair, the first member is named as e_i with concatenated string "State"; the second member consists of at most two events succeeding e_i (the first – *Received Response,* if any; the second – *Sent Response* or *Sent Request*, if any).

The first member of the next pair is *oclVoid* (if the last analyzed event is *Sent Response)* or the name of event with concatenated string "State" (if the last analyzed event is *Sent Request*). The second member is constructed in similar manner as for the first pair. If any element of the pair is missing, then *oclVoid* is used instead. The last element of the last pair is *oclVoid.*

The result of the first stage $\{\{e_1; e_2\};\{e_3; e_4\}\ldots\{e_{n-1};e_n\}\}$ is used in the second stage – generation of the corresponding state machine. The work of the algorithm is illustrated in Figures 12 and 13.

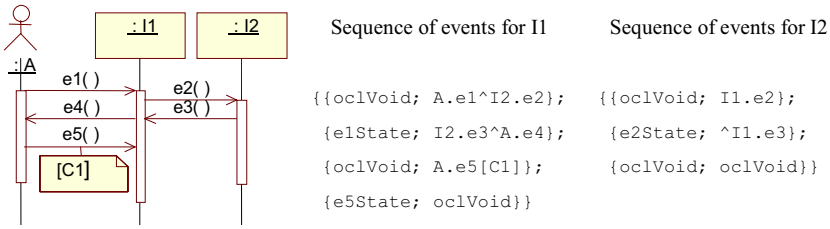


Figure 12. Example of sequences of events obtained from sequence diagram

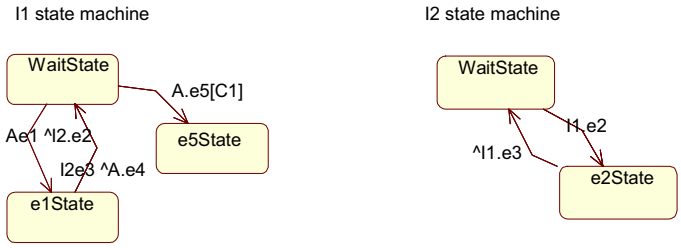


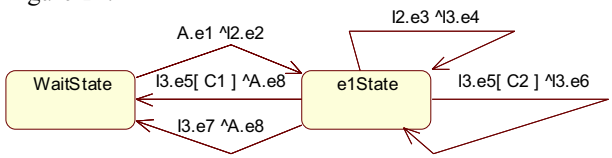
Figure 13. Resulting state machines for I1, I2

3.2. From state charts to sequence diagrams

The algorithm for generation of sequence diagrams from state charts should produce a set of sequence diagrams where every sequence corresponds to one sequence of events from state chart. Sequence is created by walking through the state chart from *Wait State* to final state and remembering the path. In one sequence each transition can be taken not more than once (this helps to avoid endless traces). Transition may be included in sequence if its source state is included in sequence and it does not have received events dependent on events that are not included in

preceding sequence (*Event e1* depends on event *e2*, if it is *Received Response* to *Sent Request e2*: *e2.Response.responseTo = e1)*.

Thus the first stage of this algorithm is to obtain all sequences from a state chart. The sequences are described in the same manner as in generation state charts from sequence diagrams. Two traces obtained from the state chart diagram are represented on Figure 14:




```
e3.Response.responseTo = e2 and e5.Response.responseTo = e4 and
                         e7.Response.responseTo = e6
{{oclVoid;   A.e1^I2.e2}                {{oclVoid;   A.e1^I2.e2}
 {e1State;   I2.e3^I3.e4}                {e1State;   I2.e3^I3.e4}
 {e1State;   I3.e5[C1]^A.e8}             {e1State;   I3.e5[C2]^I3.e6}
 {oclVoid; oclVoid}}                     {e1State;   I3.e7^A.e8}
                                         {oclVoid; oclVoid}}
```

Figure 14. Example state chart and sequences obtained from this statechart

For receiving sequence diagram pairs are analyzed according to their sequence:

- If *Lifeline* of the state machine owner, sender or receiver does not exist in the sequence diagram, it is added;
- If the first member of a pair is *Wait State*, the second member is mapped to *Request* message, received by state machine owner; *Sent Event* of this second member is mapped to *Request* message, sent by the state machine owner;
- If the first member of a pair is not *Wait State*, but the first member of subsequent pair is *Wait State*, the second member is mapped to *Response Message* and *Sent Event*, if any, is mapped to *Response Message*, sent by state machine owner to *Lifeline* of receiver of *Sent Event*;
- If the first member of a pair and the first member of subsequent pair is not *Wait State*, the second member is mapped to *Response Message*, sent to state machine owner from *Received Event* sender; *Sent Event*, if any, is mapped to *Request Message mq*, sent to *mq* receiver;
- Constraints are added as *Event Constraints* (Fig.15).

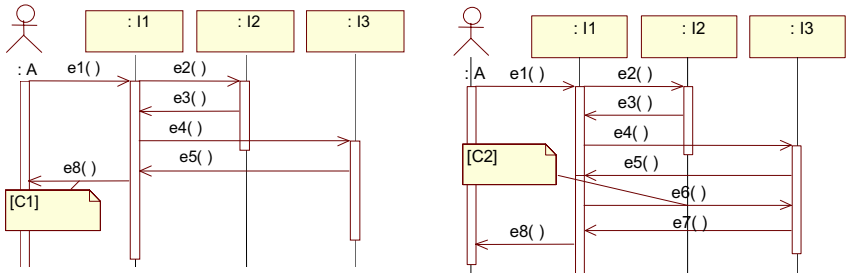


Figure 15. Sequence diagrams, derived from state machine on Figure 14

Reasoning about correctness of state machine gives possibility to improve behavior model, described by sequence diagrams. None of interactions or state related behavior definitions, presented on Figures 7–15 (except Fig.9), are complete. To ensure completeness, some additional constraints must be added to DIM metamodel, namely:

- For every synchronous *Sent Request* message at least two responsive transitions must be provided: one for the case, when response is received, and the other one, when response is not received, i.e., `e1` and `C = not(e1)` (such condition may express timeout for waiting of event occurrence);
- For every *Received Response* several conditions may hold, and several transitions or several sent messages must be provided in correspondence with these conditions.

In the last example there are lacking messages and transitions for the cases, when *Responses* are not received from *I2* or *I3* (Fig.16).

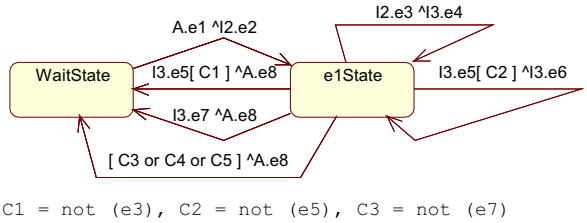


Figure 16. Improved state chart diagram

Additionally, different messages may emerge due to conditions originated from received responses. For example, after message *e3* received from *I2* at least two different messages may succeed: if *e3* conveys acceptance, request *e4* must be sent to *I2*, and if *e3* conveys rejection, response *e8* must be sent to *actor*. In general case, a set of transitions (with events and conditions) from one *Source State* must comprise tautology, and messages (with events and conditions) sent after one *Received Response* event also must comprise tautology.

In UML 2.0, the new elements – *Combined Fragments* – are proposed to use in sequence diagrams; these elements may ensure some canonical form of sequence diagram, in which all sequences for one initial *Received Request* may be represented. Also, the canonical form of state chart diagram for requirements definition may be set, where all above mentioned requirements should be ensured.

4. Design independent model example

In this section the proposed method is illustrated by fragments of DIM for IS of Publications Agency. Use Case diagram of the system is presented in Figure 17. The resulting interface class diagram is presented in Figure 18. Relationships between states of interfaces and states of entities are illustrated in Figure 19.

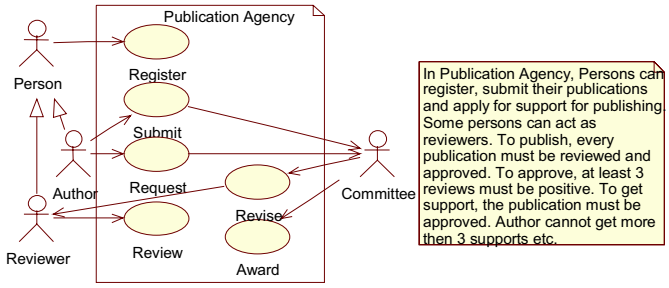


Figure 17. Use case diagram of Publication Agency

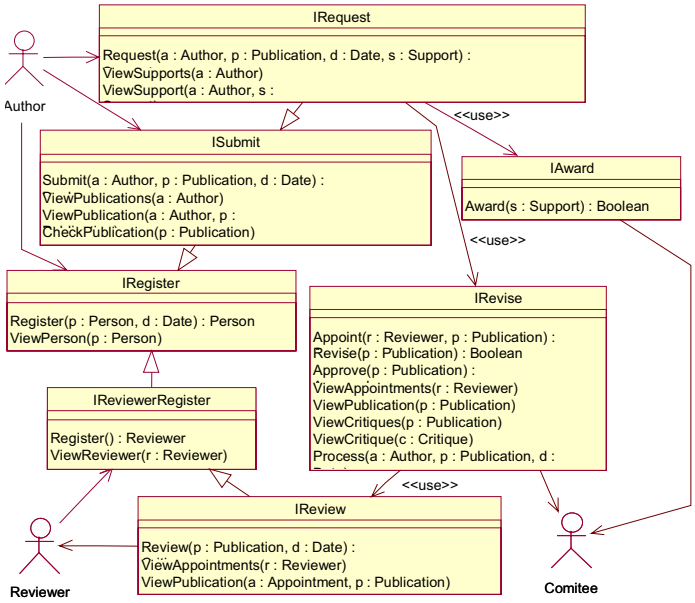


Figure 18. Representation of interfaces in class diagram

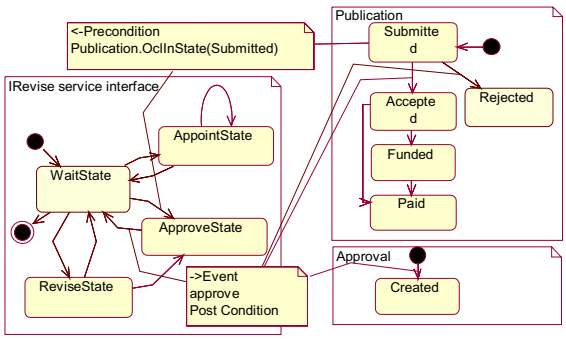


Figure 19. Relationships between states of interfaces and states of entities

5. Conclusion

Presented method for modeling of requirements to information systems consisting of services is devoted for making precise model in the sense that defined features of system should be understood unambiguously. Method is based on behavioral model where service by default is in wait state and can move to service state, during which service is provided, possibly in collaboration with other services or systems. How services are provided it is not considered in Design Independent Modeling – the main purpose is to define all actors (including external systems), interfaces, their operations, entities of problem domain, their relationships and constraints.

The idea of this work is in consequent derivation of model, fully representing design independent state and behavior of intended information system, from initial requirements. Use cases are formalized as interfaces; elaborated requirements may be presented in OCL or in visual form. Diagrams, supported with OCL constraints, are integrated and derivable from other ones. It is demonstrated, how direct and reverse mappings between sequence diagrams and state charts may be fulfilled and how analysis of state charts may reveal incomplete and inconsistent pieces.

When describing state charts, different authors prefer to use states of entities or states of actions. Really, both kinds of states are important and must be modeled in integrated manner. The form of state machines used in this paper has advantage against other kinds of state machines when service systems are modeled, as their semantics are close.

All reasoning is based on UML object model itself, on purpose to employ potential of this modeling language. In future work, it is intended to make the second step – from requirements to design, during which operations of interfaces must be allocated to control classes or components according rules of chosen design method.

REFERENCES

[1] Astesiano E., Reggio G. Knowledge Structuring and Representation in Requirement Specification. Technical Report. DISI-Universita di Genova, Italy, 2002.

[2] Bock C. Goal-Driven Modeling. Journal of Object-Oriented Programming 13(5), SIG Publications, 2000.

[3] Bock C. Three kinds of Behavior Model. Journal of Object-Oriented Programming 12(4), SIG Publications, 1999.

[4] Bordeleau F. and Corriveau J.-P.: From Scenarios to Hierarchical State Machines: A Pattern-Based Approach, in Proc. of OOPSLA 2000 Workshop: Scenario-based round-trip engineering, Tampere University of Technology, Software Systems Laboratory, Report 20, October 2000, 13-18.

[5] Ceponiene L., Nemuraite L., Paradauskas B. Design of schemas of state and behaviour for emerging information systems. Emerging Database Research in Eastern Europe. Proceedings of the Pre-Conference Workshop of VLDB 2003. Brandenburg University of Technology at Cottbus, November 2003, p. 27-31

[6] Cheesman, J. Daniels J. UML Components. Adison Wesley, 2000

[7] Cook S. et all. The Amsterdam Manifesto on OCL. Lecture Notes in Computer Science, 2263, Springer-Verlag, 2002, pp.115–149.

[8] D'Souza D. F., Wills A. C. Objects, Components, and Frameworks with UML. The Catalysis approach. Addison Wesley, 1999.
[9] Frankel D. Model Driven Architecture: Applying MDA to Enterprise Computing. John Wiley & Sons, 2003.
[10] ISO/TC97/SC5/WG3. Concepts and Terminology for the Conceptual Schema and Information Base, 1982.
[11] Jacobson I., Booch G., Rumbaugh J. The Unified Modeling Language User Guide. Addison Wesley, Boston (2000)
[12] Kleppe A., Warmer J. Extending OCL to Include Actions. Lecture Notes in Computer Science, 1939, pp. 440–450, 2000.
[13] Kosters G., Six H.W., Winter M. Coupling Use Cases and Class Models as a Means for Validation and Verification of Requirements Specifications. Requirements Engineering, 6, Springer-Verlag, 2001, pp. 3-17.
[14] Mäkinen E. and Systä T.: An Interactive approach for synthesizing UML statechart diagrams from Sequence Diagrams, in Proc. of OOPSLA 2000 Workshop: Scenario-based round-trip engineering, Tampere University of Technology, Software Systems Laboratory, Report 20, October 2000, 7-12.
[15] Martin R.C. Agile Software Development, Principles, Patterns, and Practices. Prentice Hall, 2002.
[16] Mellor S.J., Balcer M.J. Executable UML. A foundation for model-driven architecture. Addison-Wesley, 2002.
[17] Nemuraite L.,Kavaliauskaite L., Ambrazevicius E.: Towards Ensuring Consistency to UML Models. Informacijos mokslai, 24, Vilniaus universitetas, 2002, pp. 85-97.
[18] Paradauskas B., Nemuraite L. Data Bases and Semantic Models. Monograph. Technologija, Kaunas, 2002 (in Lithuanian).
[19] Reggio G., Cerioli M., Astesiano E. An Algebraic Semantics of UML Supporting it's Multiview Approach. Lecture Notes in Computer Science, 2029, Springer Verlag, Berlin, 2001.
[20] Rosenberg D., Scott K. Applying Use Case Driven Object Modeling with UML: An Annotated e-Commerce Example. – Boston: Addison Wesley, 2001. – 153 p.
[21] Sendall S., Strohmeir A. From Use Cases to System Operation specifications. <<UML>> 2000, Lecture Notes in Computer Science, 1939, Springer-Verlag, 2000, pp. 1–15.
[22] Siegel J. Developing in OMG's Model-Driven Architecture. OMG document omg/01-12-01. http://www.omg.org.
[23] Starr L. Executable UML. How to build class models. Prentice Hall, 2002.
[24] UML Profile for Enterprise Distributed Object Computing Specification. OMG document ptc/o2-02-05. 2002. http://www.omg.org.
[25] Unified Modeling Language. Superstructure Specification. Version 2.0. OMG document ptc/03-08-02, 2003. http://www.omg.org.
[26] Unified Modeling Language: OCL. Version 2.0. OMG document ptc/03-08-08, 2003. http://www.omg.org
[27] Warmer J.B., Kleppe A.G. The object constraint language: precise modeling with UML. Addison Wesley, 2000.
[28] Web Services Architecture. Working Draft. W3 Consortium, 2002. http:// www.w3.org.
[29] Whittle J., Schumann J. Generating Statechart Designs From Scenarios. International Conference on Software Engineering, Ireland, 2000, pp.314-323.

SCIENTIFIC PAPERS UNIVERSITY OF LATVIA. 2004. VOLUME 672, Datbases and Information Systems
373–384 P.

Transformation of business rules models in information systems development process

Sergejus Sosunovas, Olegas Vasilecas

Information Systems Laboratory, Vilnius Gediminas Technical University
Vilnius, Lithuania E-mail:
{sergejus, olegas}@isl.vtu.lt

Abstract. Object management group's offered model driven architecture is actively propagated to use in development of information systems. Model driven architecture enables transformation of models of business systems to the models of information systems and then to the models of software systems. Despite of numerous publications on this topic, the transformation of business rules between different enterprise systems components, for example business systems and information systems, is not addressed suitably. This paper analyses transformation of business rules, which are formally specified in UML/OCL to the event-condition-action rules of active database management systems. Meta-models necessary for the transformation and transformation problems are analysed. Paper presents results of transforming of one kind of business rules expressed in OCL to event-condition-action rules allowed to emphasize strength and weaknesses of model driven architecture framework

Keywords. Model driven architecture, meta-modelling, business rules, transformations, OCL, platform independent models, ECA rules.

1. Introduction

An unavoidable technological change challenges nowadays enterprises. In order to stay profitable and competitive enterprises have to accept this challenge and to adopt new technologies in their business environment as fast as possible. However utilization of new technology does not always mean change of the way business goes. Technological changes may result in increasing the speed of business or the quality of information what affects the quality of business decision making. The main business practices and experience possibly will stay the same.

Enterprise transition to the modern information processing and management requires to establish a new project, and to execute appropriate phases like business analysis and requirement elicitation. Bigger part of the enterprises already have executed the number of business analysis and requirement specification phases in the previous projects hence the new one project could use already created business models and requirements specifications. Nevertheless existing models were created with the old technology in mind and could not be very useful. Hence existence of technological independent business oriented models will simplify enterprise adaptation for the future technologies.

The problem of technological change and reuse of models is addressed by the object management's group (OMG) model driven architecture (MDA). MDA is based on the existence of the several layers of models and transformation of models of platform independent layer to the models of platform specific layer. At the last layer applications are generated. Model transformation is defined by transformation rules of the transformation specification. These transformation rules refer to the meta-models of model being used. Meta-models and common meta-model creation requirements defined by OMG meta object facility (MOF) and formal specification of business rules using UML/OCL [1], [2] enables model transformation in process of enterprise systems development using business rules (BR) approach.

A modern enterprise system consists of business, information and software systems [3]. It is usual to create models of business systems, elicit business needs, specify information system (IS) requirements, and then using IS needs to specify software systems requirements. At each of before mentioned steps some kind of models should be created. Transformation between these models usually is executed manually by appropriate IT staff. It is feasible to map the models of these systems to the corresponding MDA levels and automate transformation.

BR are the general part of all these systems. BR in the business systems context are defined as a directive, intended to influence or guide business behaviour, in support of business policy that has been formulated in response to an opportunity, threat, strength, or weakness [4]. Whereas in the IS context BR is statement that defines or constrains some aspect of the business [4]. Usually BR are implemented in the software code. It is possible to claim that automatic transformation of BR from one system to another will facilitate system development.

The rest of the paper proceeds as follows. Section 2 maps business, information and software system models to the corresponding MDA levels. Section 3 presents theoretical foundations of BR in IS specified in OCL transformation to the event-condition-action (ECA) rules of software systems using MDA. Transformation example of one of BR type and supporting meta-models are described in Section 4. Section 5 provides formally specified transformation rules. Section 6 summarises the proposed approach.

2. Related work

System development using MDA framework implies creation of models of following types: platform independent models (PIM) and platform specific model (PSM). Models of PIM enable specification of the several, usually selected by the user, platform independent and technological free perspectives of the modelled system. To simplify the complexity of PIM for different user groups (e.g. business staff, IT staff) several abstraction levels can be outlined: computation independent models (CIM) and PIM per se [5]. Though CIM and PIM overlap to some extent, the boundaries of these levels are defined in terms of problems and solutions they represent. CIM assumes existence of problem statement and its description, as far as PIM includes a number of platform independent solutions [6].

Informational needs and related problems are identified during analyses of business systems. Business system - the place where the problems reside, is a set of

interrelated business processes performed to achieve certain long duration goals [3]. The usage of CIM to model business systems simplifies understanding of these systems, elimination of the problems and its sources. Supporting processes of IS fulfilling the needs of business systems. IS concentrate on information and information processing methods therefore these systems are technology free and can be modelled by PIM. IS are supported by business software systems, the latter ones employs the power of different technologies to provide information processing facilities for IS. The technological solutions of business software systems are modelled by PSM.

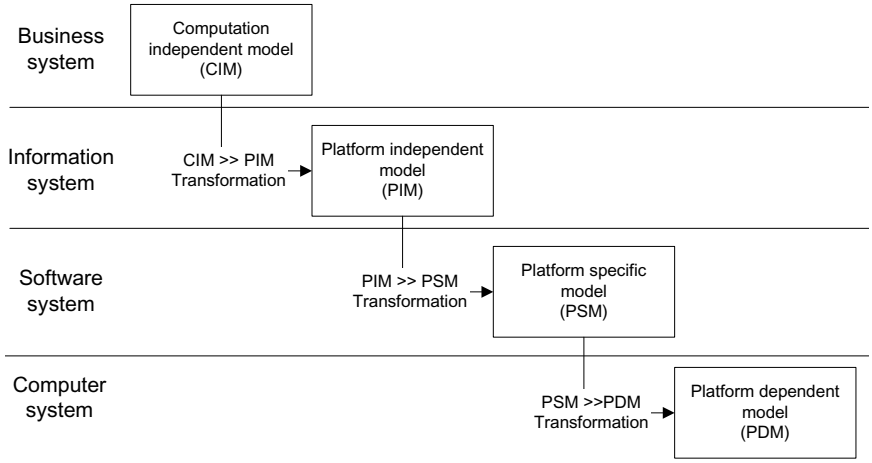


Figure 1. Mapping different levels of MDA to the different systems.

Technical environment enabling an execution of the code of business software systems itself is a system. This environment – computer system – includes necessary hardware, networks, system and middleware software, and a realisation of business software system. Computer system can be modelled using platform dependent model introduced in [6]. Mapping between presented systems and MDA parts is depicted in Fig. 1.

BR are one of the integral part of the business system. BR exist and are used in each of before mentioned systems however the form of BR may be different in every particular system. In the business system BR exists in the form of mission statement and business goals usually expressed in natural language. Besides the number of rules exist in unexpressed yet form.

BR modelling in business systems is challenging issue because of a business variety and the lack of general business vocabulary. Following an industry demand for general business vocabulary and BR meta-model OMG issued request for proposals for business semantics and BR [7]. Submitted responses [8], [9], [10] present three different approaches for BR modelling at CIM level. Meta-models for a limited number of BR are presented in [11], [12]. Additionally, BR can be specified using RuleML [13] language. The degree BR are involved in business, information and business software systems development may be different. Business rules approach [14], [15] assumes that BR are the main component of the development of these systems and provides some BR modelling methods.

BR in IS appear to be more information-centric and usually refer to some data items. BR from business system have to be transformed to corresponding constructs of IS, hence CIM have to be transformed to PIM. During this transformation the form of BR changes and additional semantic value is gained. Usually during transformation of BR specified in business system in natural language are formalised. In the context of PIM BR are most often formally expressed in OCL. This language was created to specify business constraints at IBM insurance and is powerfully enough to express different types of BR [16], [17], and [18].

OCL is declarative language however in order to implement BR on some platform it is needed to transform them to imperative form. The notion of event, when violation of BR occurs and an action is important to model BR using PSM. ECA rules are widely used in implementations of different platforms and it is feasible to transform BR specified in OCL to the ECA rules. ECA model as the result of transformation can be used to generate applications or triggers of active database management system that is more effective than implementation by means of declarative assertions [16].

Model transformation is essential part of the MDA framework. Transformation specification, a set of transformation rules, is used to define transformation of one model to another model, assuming that models are based on the meta-models complying with the MOF. Meta-model transformation approaches [19], [20] are implemented in a number of transformation specification languages: UMLX [21], ATL [21], BOTL [23], TRL [24]. Despite of numerous publications on this topic, the transformation of BR between different enterprise systems components, for example business systems and IS, is not addressed suitably.

3. Transformation decisions

OCL became part of the UML specification from version 1.1. It was developed in the IBM and is based on the first order logic. OCL is designed to specify constraints on object-oriented models (e.g UML). The semantics of UML itself is defined using OCL.

Every OCL expression begins specifying its applying context. The context may be a UML model classifier (e.g. class name, class attribute) or a method. Depending on the context it is possible to use an invariant, a keyword: inv, when context is classifier or pre- and post condition, the keywords pre and post, when the context is a method. There is a condition at the end of each OCL expression. The invariant condition must be satisfied by each instance of the model. When the context is method pre condition must be satisfied before executing the method, post condition must be true just before the end of the execution. Despite of the word “constraint” in the title of the OCL not only constraint BR can be specified. The UML 2.0 specification will include the notion of actions which will expand the OCL application domain for other BR types. One of the possible appliances of the action clause is to specify events being sent during the execution of the method, allowing to specify actions of BR, or to specify what would happen when the invariant fails.

In this paper it is assumed that constraint BR are specified using OCL invariants. OCL invariants are designated to constraint UML class diagrams. Events, initiating

part of ECA rules, are generated after some action executed on data item. In order to transform OCL clauses to the ECA rules it is necessary to transform UML class diagram to the database schema. In this paper "one-class-one-table" approach is used, each UML class is transformed to the table of relational database (RDB). Corresponding transformation of interclass relations is executed. The formal transformation specification is presented in [24]. However MDA allows easily change UML-RDB transformation principles do not affecting OCL-ECA transformation.

OCL is a purely declarative language and can be expressed in database in the following way: using assertion statements [25], using views [26]. In order to transform OCL clauses to the ECA rules it is necessary to answer the following questions [27]:

- What event initiates a trigger?
- What is trigger granularity?
- What is trigger condition?
- What is trigger initiation time?

OCL specification defines that every instance of the model must satisfy OCL constraints. Hence there are three strategies to implement OCL constraints: toleration, avoidance and combined. Constraint violation toleration strategy implies checking of all implemented OCL constraints from time to time. Usually user initiates checking and takes appropriate action if the violation is detected. In violation avoidance strategy user is alerted if violation occurs, the last transaction triggered violation is cancelled. Often in the case of complicated BR user from the business environment may be not competent enough to solve the problem which initiates violation of BR. Then combined strategy should be used. According to this strategy two types of BR are distinguished. Violation of BR of first type could be solved by the user and transaction must be cancelled. Respectively violation of the BR of the second type must be only detected and transaction must be stopped for a while.

This description of OCL clauses implementation uses the violation avoidance strategy. It is necessary to watch a value of every attribute used in the OCL clause. Hence each invariant instance must be evaluated every time those attributes are created changed, deleted. Therefore triggers implementing constraint must be initiated on database column creation, modification and deletion actions.

Trigger initiation time implies the time when the trigger condition must be evaluated, before execution of trigger initiated action or after. But if the trigger condition is executed before the action it does not have access to the created/modified data. It necessary to check condition after trigger initiated action.

Inserting n records in a time trigger can be executed one or several times, depending on the trigger granularity. There are two types of trigger granularity statement level and row level. Using violation tolerance strategy, when all constraints are evaluated on the snapshot of the database it would be feasible to use statement level triggers, however avoidance strategy must be implemented using row level triggers.

OCL clauses are effect free, which means that specification doesn't describe what action should be executed if the violation of constraint occurs. Implementing OCL clauses in ECA rules of active database systems the system can just alert user about constraint violation or cancel transaction. During database transaction execution it is

4. Example of transformation

In the following example BR belonging to the comparative evaluators' family will be transformed to the ECA rules. The comparative evaluators' BR family contains business rules of several types: equal-to, not-equal-to, greater-than, greater-than-or-equal-to, less-than. An example of comparative business rule: "Cigarettes can not be sold to the customers younger than 18 years old". This business rule expressed in OCL (context is presented in Fig. 2):

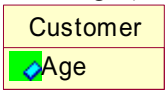


Figure 2 UML model for the OCL example.

```
Context Customer
Inv AgeConstraint: Age >=18
```

In order to transform such business rule to the ECA rule according to MDA framework two meta-models are needed, one describing the semantics of OCL and the second describing semantics of ECA rules. OCL specification already contains general meta-model, for our purposes that meta-model is too big and the view of it must be created. Simplified OCL meta-model used for transformation of comparative evaluators' BR and complete to present such kind of BR is depicted in Fig 3 and Fig. 4.

ECA meta-model which is partly based on meta-model presented in [11] and extended to include ability to represent conditional statements is depicted in Fig. 6. Event of ECA rule is usually related to some data structure, in simplified ECA meta-model this data structure is "Table" from [24] (Fig. 5).

As it is common for MOF compliant meta-models almost all elements are inherited from one single element. In ECA meta-model case this element is called "ModelElement" (Fig 7.) and its one attribute "name", used in the child elements in a number of ways (e.g. to represent operation sign, and elements names).

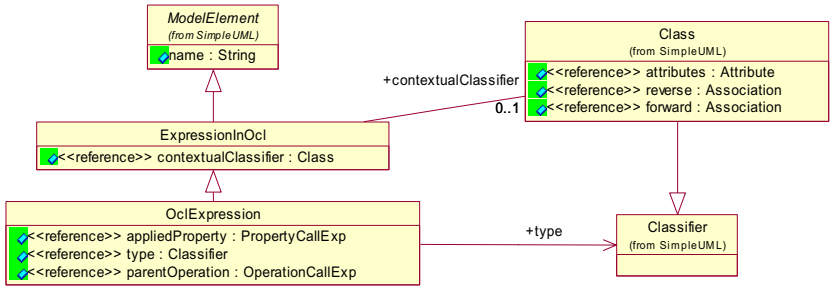


Figure 3 Simplified OCL meta-model (First part).

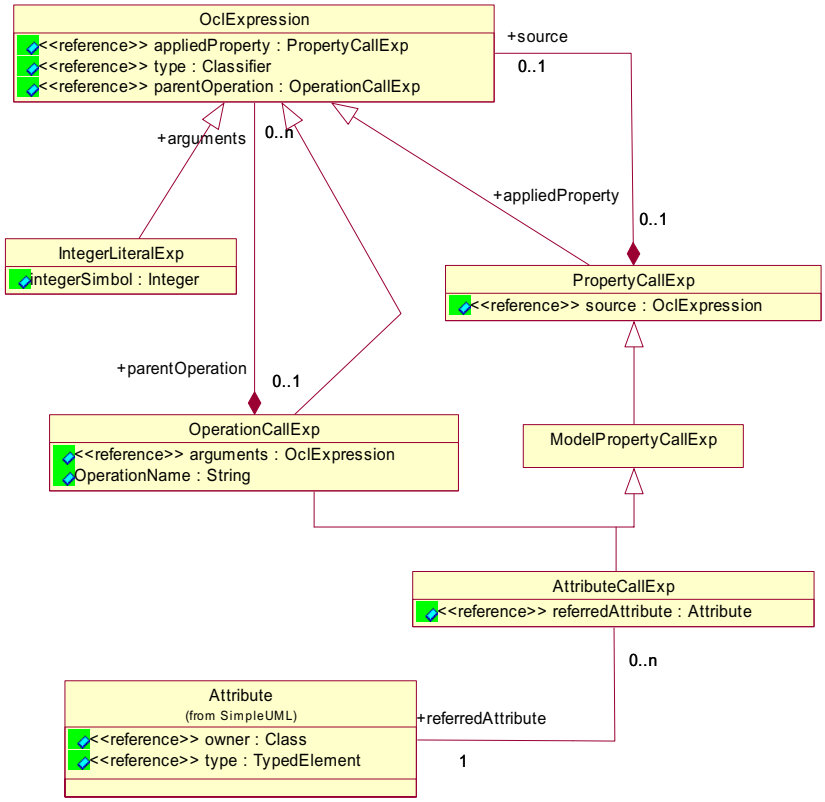


Figure 4. Simplified OCL meta-model (Second part).

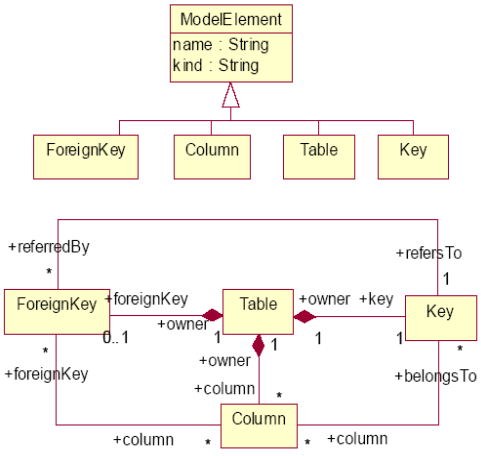


Figure 5 Simplified relational data base meta-model [24].

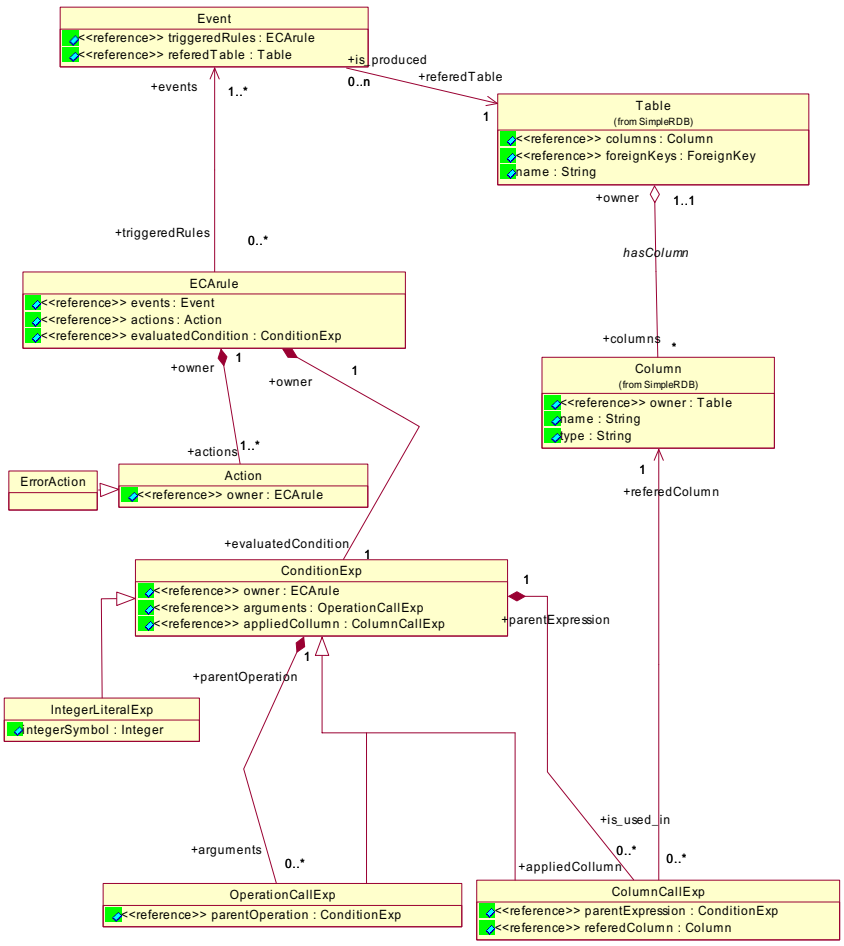


Figure 6. Simplified ECA rule meta-model.

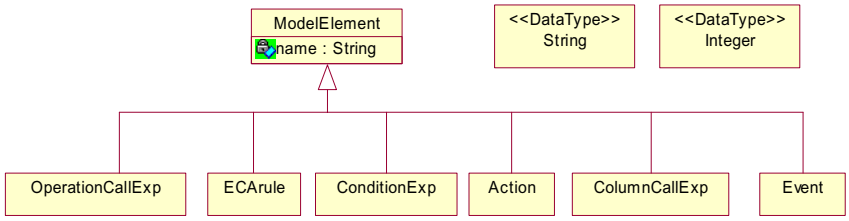


Figure 7. Classes of ECA simplified meta-model.

5. Transformation specification

There is no standard transformation language for the specification of transformation rules of meta-model based transformations. However the general structure of transformation rules proposals is quite similar. Every transformation language first declares the meta-model or the view of meta-models being used in transformation. There is a pattern in the beginning of each transformation rule specifying what source meta-model elements will be used for the transformation and destination meta-model constructs will be produced. The main differences between different transformation languages are: the type of transformation rules (declarative or imperative), the notion of source-destination models, rule usage strategies, rule organisation, transformation directions [19].

Transformation rule language (TRL) [24], is used to define model queries and transformations according to MOF 2.0 meta-model principles. This language is an OCL extension, hence bigger part of methods including model navigation methods are taken from OCL. TRL is organised in modules and packages. Three main packages are: query, transformation and view.

TRL extends OCL in a number of ways aimed to enable and facilitate transformation. Low coupling between different transformation specifications allows easily apply different transformation approaches for the different parts of the same meta-model. Ability to refer to the already transformed elements simplifies the development of specification.

OCL-ECA transformation specification presented in this paper is defined using TRL. This specification extends UML-DBMS transformation specification defined in [24]. UML-DBMS transformation specification is necessary to refer to the "Column" and "Table" already transformed elements.

OCL-ECA transformation specification (formal definition):

```
transformation Ocl2ECA

-- Reference to the already executed transformation
extends transformation Uml2rdbms;

-- Declaration of used meta-models
use modeltype SimpleOCL, SimpleECA

-- Transformation direction specification
purpose create ecamodel:SimpleECA from oclmodel:Simple:OCL

-- Activation of the transformation.
activator go()
   {oclmodel.objects() [OclExpression].create ECArule();}

-- The first rule transforming OCL expression to ECA rule
rule R1 create ECArule from OperationCallExpression(){
-- ECA rule will have the same name as OCL constraint
   name:=name;
-- The sequence of events must be generated for the each ECA rule
-- element
   events:=ContextualClassifier.create Sequence(Event);
   evaluatedConditions:=self.create OperationCallExp();
   action:=self.create ErrorAction();}

-- The ECA condition must be checked on UPDATE and INSERT events
```

```
rule R2 create Sequence(Event) from ContextualClassifier(){
   result:={self.create Event("UPDATE"),
      self.create Event("INSERT")};}

rule R3 create Event(n:String)from ContextualClassifier(){
   name:= name+"_"+n;
--Resolution of the already transformed object Table
   table:=self.resolve(Table);}

rule R4 create OperationCallExp() from OperationCallExp(){
   name:= "not "+OperationName;
   owner := self.resolve(ECArule);
   arguments := arguments.Create Sequence(ConditionExp)
                  from OclExpression;
   }

rule R5 create Sequence(ConditionExp) from OclExpression
{
   -- Here the OCL if expressions and IsTypeOf operation are used to
   -- transforms necessary arguments
     :result = if self.IsTypeOf(IntegerLiteralExp)
             then self.create IntegerLiteralExp().asSequence()
             else if self.IsTypeOf(AttributeCallExp)
               then self.create ColumnCallExp().asSequence()
              end
             end
}

rule R6 create IntegerLiteralExp from IntegerLiteralExp
{
   name:=name;
   integerSymbol := integerSymbol;
}

rule R7 create ColumnCallExp from AttributeCallExp
{
   name:=name;
   referredColumn := self.referredAttribute.resolve(Column);
}

rule R7 create ErrorAction() from ExpressionInOcl()
{
   owner := self.resolve(ECArule);
   name := "Violation of: "+name;
}
```

6. Conclusions

Transformation development process has some similarities with classical system development processes. First the transformation requirements are specified, then transformation is designed in a declarative manner. At the high abstract level the mapping between appropriate components of destination and source models is defined. Transformation is implemented specifying imperative transformation rules. In order to achieve necessary quality the transformation correctness should be evaluated. It is feasible to apply iterative transformation development process.

Developing of the transformation rules is very close to the programming of usual software. So well known approaches to the organisation of business rules can be applied. However because of the specificity of transformation rules (e.g. high granularity) the new programming techniques and tools should be developed.

MDA enables uncomplicated migration of the BR based systems to the new platforms. Modelling of BR in platform independent language OCL allows precise and unambiguous specification of BR. Presented transformation of OCL clauses to ECA rules and then to trigger of active DBMS is a common and efficient approach to implement BR. BR implemented as triggers are more easily maintained and changed.

References

[1] OMG UML 2.0 OCL Draft Adopted Specification. OMG document: ptc/03-08-08. http://www.omg.org/cgi-bin/apps/do_doc?ptc/03-08-08.pdf

[2] OMG. OMG/MOF Meta Object Facility (MOF) Specification. OMG document: formal/02-04-03, http://www.omg.org/cgi-bin/apps/do_doc?formal/02-04-03.pdf.

[3] Ļaplinskas, A., Lupeikienā, A., Vasilecas, O. Shared Conceptualisation of Business Systems, Information Systems and Supporting Software. Databases and Information Systems II. Kluwer Academic Publishers, 2002, pp. 109-120.

[4] Ross, R., G. Principles of the Business Rule Approach. Addison Wesley, 2003.

[5] Siegel, J. Developing in OMG's Model-Driven Architecture. OMG document: 01-12-01, 2001. http://www.omg.org

[6] Bézivin, J., Gerard. S., Muller, P.-A., Rioux L. MDA components: Challenges and opportunities. Proceedings of Metamodelling for MDA workshop, York, England, November, 2003. http://www.cs.york.ac.uk/metamodel4mda/onlineProceedingsFinal.pdf

[7] OMG. Business Semantics of Business Rules RFP. OMG document: br/03-06-03, 2003. http://www.omg.org/cgi-bin/apps/do_doc?br/03-06-03.pdf.

[8] Adaptive, Business rule Solutions LLC et all. Business Semantics of Business Rules. OMG document: bei/2004-01-04, 2004. http://www.omg.org/cgi-bin/doc?bei/04-01-04

[9] Fujitsu Limited. Business Semantics of Business Rules. OMG document: br/2004-01-01, 2004. http://www.omg.org/cgi-bin/doc?br/04-01-01

[10] International Business Machines. Business Semantics of Business Rules. OMG document: br/01-01-02, 2004. http://www.omg.org/cgi-bin/doc?br/04-01-02

[11] Herbst, H. A Meta-Model for Business Rules in Systems Analysis. CAiSE'95 proceeding. Berlin, Springer, 1995, pp. 186-199.

[12] Moriarty, T. Business Rule Management Facility: System Architect 2001. On-line paper. http://www.intelligententerprise.com/000801/metaprise.jhtml?_requestid=441609

[13] Boley, H. The Rule Markup Language: RDF-XML Data Model, XML Schema Hierarchy, and XSL Transformations. Proceedings of INAP2001 LNCS 2543. Invited Talk, Tokyo, Springer-Verlag, 2003, pp. 5-22.

[14] Ross, R., G. The business Rule Book (2nd ed.). Business Rule Solutions, Houston, 1997.

[15] Von Halle, B. Business rules applied: building better systems using the business rules approach. John Wiley & Sons, New York, 2001.

[16] Wagner, G., Tabet, S., Boley, H. MOF-RuleML: The Abstract Syntax of RuleML as a MOF Model. Integrate 2003 proceeding. http://www.omg.org/docs/br/03-09-01.doc

[17] Sosunovas, S., Vasilecas, O., Using UML/OCL for business rules modelling. Proceedings of Sixth Lithuanian young scientist conference, Vilnius Gediminas Technical University, Technika, Vilnius, 2003, pp: 139-148.

[18] Sosunovas, S., Vasilecas, O. The meta-model based transformation of business rules models. In A. Caplinskas, V. Denisov at al (eds.). Proc. of the Conference "Information Technology'2004", Kaunas University of Technology, 2004, pp. 585-592. (in Lithuanian)
[19] Czarnecki, K., Helsen., S. Classification of Model Transformation Approaches. OOPSLA'03 Workshop on Generative Techniques in the Context of the MDA proceedings, 2003, http://www.softmetaware.com/oopsla2003/mda-wokshop.html
[20] Gardner T., Griffin C., Koehler J., Hauser R. A review of OMG MOF 2.0 Query/View/Transformations toward the final Standard. OMG document: ad/03-08-02. http://www.omg.org/cgi-bin/apps/do_doc?ad/03-08-02.pdf
[21] Willink, E. D. UMLX - A Graphical Transformation Language for MDA, Workshop on Model Driven Architecture Foundations and Applications, University of Twente, The Netherlands, 2003, pp. 13-24.
[22] Bézivin, J., Dupé, G., Jouault, F., Pitette, G., Rougui, J., E., First experiments with the ATL model transformation language: Transforming XSLT into XQuery, 2nd OOPSLA Workshop on Generative Techniques in the context of Model Driven Architecture, October 23-30 2003, California, USA. http://www.softmetaware.com/oopsla2003/bezivin.pdf
[23] Brauna, P., Marschall, F. The Bi-directional Object-oriented Transformation Language. Technical report, Technische Universität München, TUM-I0307, 2003, http://wwwbib.informatik.tu-muenchen.de/infberichte/2003/TUM-I0307.pdf
[24] Alcatel, Softeam, Thales, TNI-Valiosys, Codagen Technologies Corp. Response to the MOF 2.0 Query/View/Transformations RFP. OMG document: ad/2003-08-05, 2003, http://www.omg.org.
[25] Demuth, B., Hussmann. H., Using OCL constraints for relational Database Design. LNCS 1723, Springer-Verlag Berlin Heidelberg, 1999, pp. 598-613.
[26] Demuth, B., Hussmann, H., Loecher. S. OCL as a Specification Language for Business rules in Database Applications Design. LNCS 2185, Springer-Verlag Berlin Heidelberg, 2001, pp. 104-117.
[27] Badawy, M., Richta. K. Deriving triggers from UML/OCL specification. In M. Kirikova at al (Eds.). Proceedings of ISD 2002 conference on Information Systems Development: Advances in Methodologies, Components and Management. Kluwer Academic/Plenum Publishers, 2002, pp. 305-316.

SCIENTIFIC PAPERS UNIVERSITY OF LATVIA. 2004. VOLUME 672, Datbases and Information Systems
385–400 P.

Reverse Engineering of Relational Databases to Object Databases

Irina Astrova

Tallinn University of Technology
Ehitajate tee 5, 19086 Tallinn, Estonia
irina.astrova@cellnetwork.com

Abstract. Whereas reverse engineering of relational databases to object databases is a widely studied subject, a majority of the work on schema transformation has been done on analyzing key correlation. Data and attribute correlations are considered rarely and thus, have received little or no analysis. This is partially evident from that the existent approaches can extract only a small subset of semantics embedded within a relational database; for example, they can fail in discovering inheritance and optimization structures. Another problem with the existing approaches is that they can require much user interaction for semantic interpretation, thus giving less opportunity for automation. As an attempt to resolve these problems, we propose a novel approach, which is based on an analysis of key, data and attribute correlations, as well as their combination. Our approach transforms a relational database schema into an object database schema in the context of Unified Modeling Language (UML). It has two important advantages over the existing approaches in that: (1) it allows the user to extract more semantics from a relational database, and (2) it reduces user interaction, which is mainly required to confirm the extracted semantics and to give them names.

Keywords. Schema transformation, relational databases, object databases

1. Introduction

Reverse engineering of relational databases to object databases is defined as a process that obtains semantics about the legacy database, then converts the schema from relational to object, and finally represents the results as an object database schema [1]. There are several motives that require reverse engineering of relational databases to object databases. For example:

- To gain an understanding of semantics of the legacy database [2]
- To document the legacy database structure (i.e. schema) [3]
- To migrate the database from relational to object [4].

Regardless of these motives, the main task of reverse engineering of relational databases to object databases is *schema transformation*, that is, the translation of the relational database schema into a semantically equivalent database structure expressed in the new (e.g. object) technology [5]. In particular, the schema transformation consists of deriving an object database schema from a relational database schema, by replacing a construct (possibly empty) in the relational database

schema with a new construct (possibly empty) in the object database schema [1]. Mapping a relation into a class, replacing a primary/foreign key with a generalization relationship are examples of the schema transformation.

2. Related work

Over the past ten years, researchers have proposed different approaches to reverse engineering of relational databases to object databases, as shown in Table 1. The existing approaches differ in their specific assumptions, inputs and methodological characteristics. However, a majority of that research has been done on analyzing key correlation. Data and attribute correlations are considered rarely and thus, have received little or no analysis.

Table 1. Approaches to reverse engineering of relational databases to object databases

Approach	Assumptions	Input	Output	Characteristics
Albert et al.'s [6]	3NF	- Relational database schema - Data instances	Object database schema	- Provides a simple and an automatic schema transformation - But does not cope with inheritance and optimization structures
Premerlani et al's [3]	Non-3NF	- Relational database schema - Data instances	Object database schema (in the context of OMT)	- Emphasizes on an analysis of candidate keys, rather than primary keys - Copes with optimization structures - But requires much user interaction
Ramanathan and Hodges' [7]	3NF	- Relational database schema - Key dependencies	Object database schema (in the context of OMT)	- Requires less input information - But does not cope with optimization structures
Behm et al.'s [4]	3NF	- Relational database schema - Data instances - Key dependencies - Inclusion dependencies - Types of relationships (strong or weak)	Object database schema	- Includes both schema transformation and data mapping - But considers only data equality and data inclusion

Tari and Stokes' [8]	3NF	- Relational database schema - Data instances - Key dependencies	Object database schema	- Emphasizes on an analysis of external keys, rather than primary keys - Analyzes key and data correlations, as well as their combination - But does not distinguish between data equality and data inclusion

This is partially evident from that the existent approaches can **extract only a small subset of semantics embedded within a relational database**, thus producing an object database schema that looks rather "relational". For example, [4, 6, 8, 7] fail in discovering optimization structures. Even though, more recently, new tools have appeared to provide an automatic mapping of the relational model to the object model, they still have limitations. For example, Rational Rose [9] maps an inheritance structure to a composite aggregation. After the schema transformation, the user can replace the composite aggregation with a generalization, also known as an inheritance hierarchy.

Another problem with the existing approaches is that they can **require much user interaction for semantic interpretation**, thus giving less opportunity for automation. For example, even though [3] discovers inheritance and optimization structures, it represents a fairly informal process of schema transformation with weakly ordered steps that entail a lot of involvement from the user, backtracking and reodering.

As an attempt to resolve the problems, we propose a novel approach, which is based on an analysis of key, data and attribute correlations, as well as their combination. Our approach transforms a relational database schema into an object database schema in the context of Unified Modeling Language (UML) [9, 10].

Our choice of UML yields an important advantage. One notation is used to represent both schemata. The existing approaches usually use Peter Chen's Entity-Relationship (ER) notation for relational database schema representation and the OMT (Object Modeling Technique) for object database schema representation.

3. Relational database schema vs. object database schema

Before going into detail of schema transformation, we give a formal description of: (1) relational database schema, (2) object database schema, and (3) relationship between the two.

3.1. Relational database schema

This is the source for the schema transformation. Formally, a relational database schema (based on the relational model) consists of:

- A set of relations R
- A set of attributes A_R
- A set of relational database types T_R that includes only simple types
- A function *attr*: $R \rightarrow 2^{A}{}_{R}$ that returns attributes of relations: $\forall r(r \in R \wedge attr(r) \subseteq A_R)$
- A function *type:* $A_R \rightarrow T_R$ that returns types of attributes: $\forall a(a \in A_R \wedge \exists t(t \in T_R \wedge type(a)=t))$
- A function *key:* $R \rightarrow 2^{A}{}_{R}$ that returns primary keys of relations: $\forall r(r \in R \wedge key(r) \subseteq attr(r))$ [4].

3.2. Object database schema

This is the target for the schema transformation. Formally, an object database schema (based on the object model) consists of:

- A set of classes C
- A set of attributes A_O
- A set of object database types T_O that includes both simple and composite types
- A function *attr:* $C \rightarrow 2^{A}{}_{O}$ that returns attributes of classes: $\forall c(c \in C \wedge attr(c) \subseteq A_O)$
- A function *type:* $A_O \rightarrow T_O$ that returns types of attributes: $\forall a(a \in A_O \wedge \exists t(t \in T_O \wedge type(a)=t))$ [4].

3.3. Relationship between schemata

We describe how the relational database schema is related to the object database schema using:

- A function *rel:* $C \rightarrow R$ that returns relations from which classes have been derived. Generally, each class corresponds to a single relation: $\forall c(c \in C \wedge \exists r(r \in R \wedge rel(c)=r))$
- A function *tt:* $T_R \rightarrow T_O$ that translates relational database types into object database types (e.g. VARCHAR into String), because of "impedance mismatch" between the two type systems [11]: $\forall a(a \in A_R \wedge \exists t(t \in T_O \wedge tt(type(a))=t))$ and $T_R \subset T_O$.

4. Our approach

The schema transformation uses a relational database in third normal form (3NF)[1] as the main input. It goes through five steps: (1) classification of relations, (2)

[1] The reason for starting schema transformation with 3NF relations is that such relations are the "best" structures, which reflect object concepts [8]. "More normalized" relations, such as 4NF and BCNF, may break relations at the level of loosing the original structures of classes. On the other hand, "less normalized" relations, such as 1NF and 2NF, may leave relations

mapping relations, (3) mapping attributes, (4) mapping relationships, and (5) establishing cardinalities.

We illustrate each of these steps for the schema transformation using the UML notation [9, 10].

4.1. Classification of relations

Relations are classified into one of the three categories: (1) base relations, (2) dependent relations, and (3) composite relations.

4.1.1. Base relations

If a relation is independent of any other relation in the relational database schema, it is a base relation. Formally, a relation $r \in R$ is a *base relation*, if $\neg\exists r_1 \in R$ such that $K_1 \subset K$, where $K=key(r)$ and $K_1=key(r_1)$.

In Fig. 1, `Department` is a base relation, as it has no foreign key. An attribute `Department_ID` in `Employee` is a foreign key to `Department`. However, `Employee` is also a base relation, as `Department_ID` is not part of its primary key. Yet another example of base relation is `Project`.

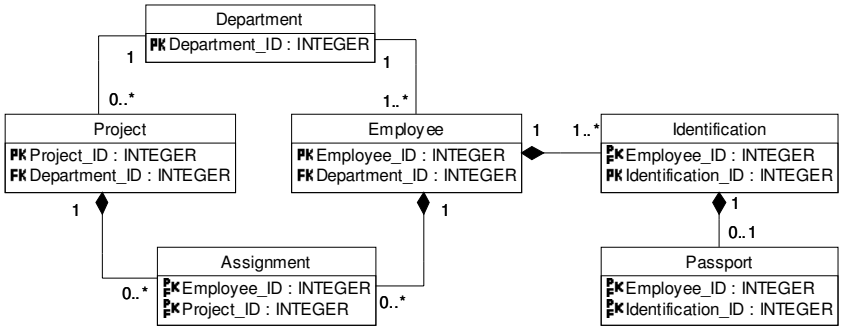


Figure 1. Classification of relations

4.1.2. Dependent relations

If a primary key of a relation depends on another relation's primary key, it is a dependent relation. Formally, a relation $r \in R$ is a *dependent relation*, if $\exists r_1, r_2 \ldots r_n \in R$ such that $K_1 \subseteq K_2 \ldots \subseteq K_n \subseteq K$, where $K=key(r)$, $K_i=key(r_i)$, $i \in \{1 \ldots n\}$ and $n \geq 2$.

In Fig. 1, `Identification` is a dependent relation, as it gets `Employee`'s `Employee_ID` that is part of its primary key.

---

containing many tuples, which are difficult to assembly into objects during the data mapping process.

4.1.3. Composite relations

All other relations fall into this category. Formally, a *composite relation* is a relation that is neither base nor dependent.

In Fig. 1, `Assignment` is a composite relation, as its primary key is composed of primary keys of `Employee` and `Project`[2].

4.2. Mapping relations

Mapping relations is straightforward and usually poses no difficulties in the schema transformation, as each relation becomes a class, with the exception of composite relations. A composite relation is difficult to map, because depending on its structure, it can correspond up to three different constructs in an object database schema:

- A binary or higher degree association, when all its attributes are primary/foreign keys[3]
- An association class, when it contains an additional attribute that is not a primary key or primary/foreign key
- A qualified association, when it contains an additional primary key.

In Fig. 2, a base relation `Employee` maps to a class `Employee`. Similarly, a base relation `Project` maps to a class `Project`. A composite relation `Assignment` becomes a binary association, as it consists entirely of primary keys of all its associated relations: `Employee` and `Project`.

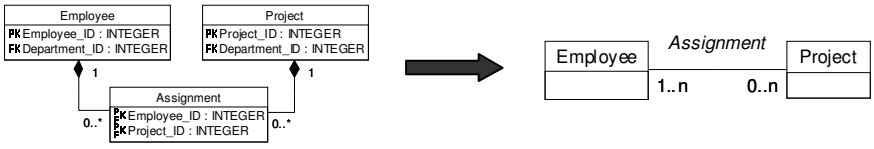


Figure 2. Mapping relations: A binary association

In Fig. 3, not only does a composite relation `Employment` comprise primary keys of all its associated relations (`Employee` and `Department`), but it also

---

[2] Because both dependent and composite relations require a composite key – the key that has more than one attribute in it – sometimes it may be difficult to distinguish between them. In Fig. 1, `Passport` "looks" like a composite relation, as its primary key is composed of primary keys of `Employee` and `Identification`:
key(`Passport`)=key(`Employee`)∪key(`Identification`).
However, it is a nesting of keys that classifies `Passport` as a dependent relation, as opposed to it being a composite relation:
key(`Employee`)⊆key(`Identification`)⊆key(`Passport`).

[3] A composite relation can also map to a generalization (namely, a multiple inheritance), when all its attributes are primary/foreign keys. But as there is no really good way to represent a multiple inheritance in a relational database schema, a composite relation will map to an association. The user can then replace that association with a generalization.

contains the date, when an employee started to work for the department. Therefore, `Employment` becomes an association class, the class that acts as an association, but yet has an attribute `startDate`.

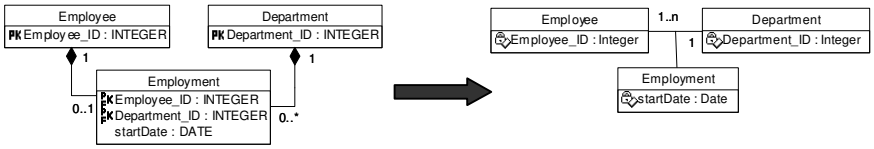


Figure 3. Mapping relations: An association class

In Fig. 4, a primary key attribute `deptName` in a composite relation `Belongs_to` is a qualifier, meaning that a company has many different departments, identified by a (unique) department name. Therefore, `Belongs_to` becomes a qualified association.

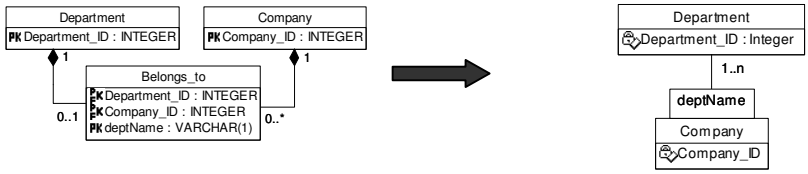


Figure 4. Mapping relations: A qualified association

4.3. Mapping attributes

Each attribute in a relation becomes an attribute in a class, with the exception of foreign keys and primary/foreign keys. A foreign key and primary/foreign key are ignored, as they refer to another relation.

In Fig. 5, we add an attribute name (of type `String`) to a class `Employee`.

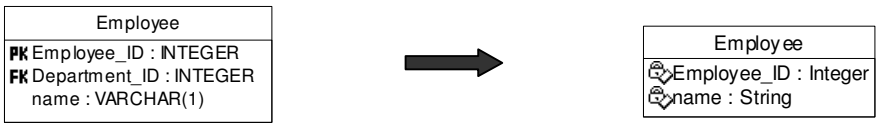


Figure 5. Mapping attributes

4.4. Mapping relationships

In the relational database schema, relationships are represented through foreign keys and primary/foreign keys[4]. For example:

- In Fig. 1, an employee *works for* a department. This relationship is represented by an attribute `Department_ID` in Employee. `Department_ID` is a foreign key to `Department`
- In Fig. 1, a passport *is* an identification document. This relationship is represented by an attribute `Identification_ID`, which appears as a primary/foreign key in `Passport`.

A foreign key and primary/foreign key can correspond to a binary association, generalization or composite aggregation, depending on the types of key, data and attribute correlations.

4.4.1. Key, data and attribute correlations

Formally, given two relations $r_1 \in R$ and $r_2 \in R$:

- The types of *key correlation*: key equality ($K_1=K_2$), key inclusion ($K_1 \subset K_2$), key overlap ($K_1 \cap K_2 \neq \varnothing$, $K_1 - K_2 \neq \varnothing$, $K_2 - K_1 \neq \varnothing$) and key disjointedness ($K_1 \cap K_2 = \varnothing$)
- The types of *data correlation*: data equality ($r_1[K_1]=r_2[K_2]$), data inclusion ($r_1[K_1] \subset r_2[K_2]$), data overlap ($r_1[K_1] \cap r_2[K_2] \neq \varnothing$, $r_1[K_1] - r_2[K_2] \neq \varnothing$, $r_2[K_2] - r_1[K_1] \neq \varnothing$) and data disjointedness ($r_1[K_1] \cap r_2[K_2] = \varnothing$)
- The types of (non-key) *attribute correlation*: attribute equality ($A_1=A_2$), attribute inclusion ($A_1 \subset A_2$), attribute overlap ($A_1 \cap A_2 \neq \varnothing$, $A_1 - A_2 \neq \varnothing$, $A_2 - A_1 \neq \varnothing$) and attribute disjointedness ($A_1 \cap A_2 = \varnothing$),

where $K_1=key(r_1)$, $K_2=key(r_2)$, $A_1=attr(r_1)-K_1$, $A_2=attr(r_2)-K_2$, $r_1[K_1]=\pi_{K1}(r_1)$ and $r_2[K_2]=\pi_{K2}(r_2)$.

4.4.2. Analysis of key correlation

To map relationships, we start with analyzing key correlation. Then we proceed on to an analysis of data and attribute correlations, because additional ("hidden") semantics, such as inheritance and optimization structures, are not solely contained in keys, but also in tuples (i.e. data) and attributes.

First, consider a relationship between `Project` and `Task` in Fig. 6, when **key inclusion** holds on it[5]. That relationship maps to a composite aggregation (also known as a non-shared aggregation, an aggregation by value or just a composition), because the identification of instances in `Task` requires the primary key of

---

[4] Relationships can also be represented through composite relations, but we have already handled them in the prior step of schema transformation (see Section 4.2). For example, in Fig. 2 an employee *is assigned to* a project. This relationship is represented through a composite relation `Assignment`. Here an attribute `Employee_ID` refers to `Employee`, while `Project_ID` to `Project`.

[5] If tasks were uniquely identified only within a project, then the primary key of `Project` (i.e. `Project_ID`) would be combined with an attribute `Task_ID` to define the primary key of `Task`: key(`Project`) ⊂ key(`Task`).

`Project` (i.e. `Project_ID`). That is, `Project` has an identifying relationship with `Task`.

Figure 6. Mapping relationships: key inclusion

Second, consider a relationship between `Project` and `Task` in Fig. 7, when **key disjointedness** holds on it[6]. That relationship maps to a binary association, as it associates two relations that are independent of each other, or use a non-identifying relationship. (Indeed, the relationship cannot be identifying, because the identification of instances in `Task` does not require any attribute in `Project`'s `Project_ID`.)

Figure 7. Mapping relationships: key disjointedness

4.4.3. Analysis of data and attribute correlations

In mapping relationships, while the analysis of key correlation is crucial for deriving semantics from the relational database schema, data and attribute correlations are also important to analyze. For example, key equality can determine an inheritance structure. But the type of inheritance (either single or multiple) is determined with the types of data and attribute correlations. The importance of the analysis of data and attribute correlations is perhaps best explained by example.

First, consider a relationship between `SoftwareProject` and `Project` in Fig. 8, when **key equality**, **data equality** and **attribute disjointedness**[7] hold on it. This is an example of vertical partitioning, where attributes of a single (logical) relation have been split into two relations, having the same primary key. Therefore, we combine `SoftwareProject` and `Project` into a single class, say `SoftwareProject`, whose attributes are the union of the attributes of the two relations.

---

[6] If an attribute `Task_ID` in `Task` were always unique, even between projects, then `Task` would get `Project`'s `Project_ID` that was not part of its primary key: key(`Project`)∩key(`Task`)=∅.

[7] By attribute "disjointedness", we mean that the two relations have no common attributes other than primary keys, foreign keys and primary keys/foreign keys.

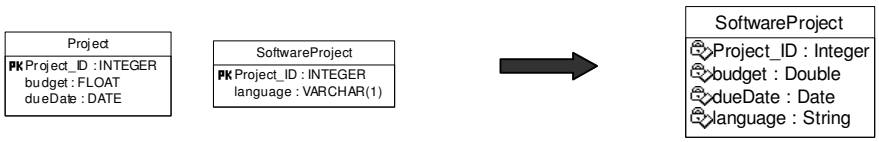


Figure 8. Mapping relationships: key equality, data equality and attribute disjointedness

Second, consider a relationship between `SoftwareProject` and `Project` in Fig. 9, when **key equality, data disjointedness** and **attribute equality** hold on it. This is, again, an example of optimization structure; but horizontal partitioning, where data of a single (logical) relation have been split into two relations, having the same attributes. Therefore, we combine `SoftwareProject` and `Project` into a single class, say `SoftwareProject`, whose data are the union of the data of the two relations.

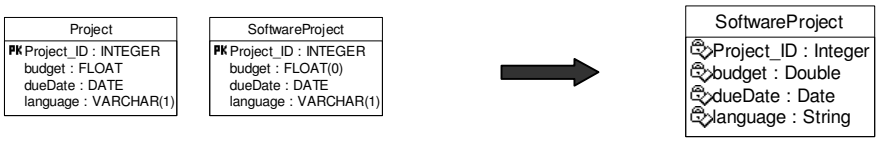


Figure 9. Mapping relationships: key equality, data disjointedness and attribute equality

Third, consider a relationship between `SoftwareProject` and `Project` in Fig. 10, when **key equality** and **data inclusion** hold on it. That relationship maps to a single inheritance, as all data of `SoftwareProject` are also included in `Project`; i.e., a software project is a project. But the converse is not true, as some projects can be hardware projects, for example.

Figure 10. Mapping relationships: key equality and data inclusion

Fourth, consider a relationship between `SoftwareProject` and `HardwareProject` in Fig. 11, when **key equality, data overlap** and **attribute disjointedness** hold on it. This is an example of a multiple inheritance, as some data are common to both relations. Therefore, we "discover" a new class, say `HardwareSoftwareProject`, which has generalization relationships to both `SoftwareProject` and `HardwareProject`.

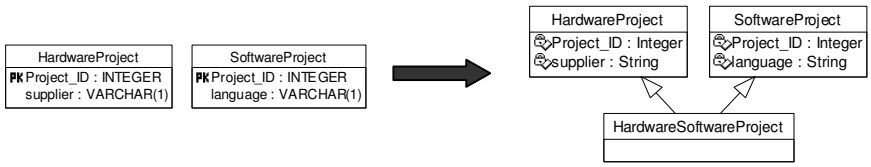


Figure 11. Mapping relationships: key equality, data overlap and attribute disjointedness

Fifth, consider a relationship between `SoftwareProject` and `HardwareProject` in Fig. 12, when **key equality, data disjointedness** and **attribute overlap** hold on it. This example also illustrates generalizations, but a single inheritance. Because some attributes (e.g. `budget` and `dueDate`) are common to both relations, we can see that `SoftwareProject` and `HardwareProject` are part of the inheritance hierarchy, but there is no relation corresponding to their superclass. Therefore, we "discover" a new class, say `Project`, which both `SoftwareProject` and `HardwareProject` inherit from.

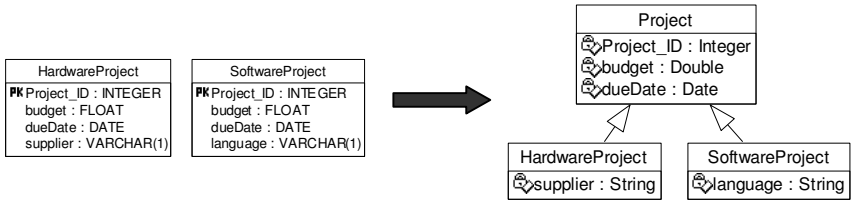


Figure 12. Mapping relationships: key equality, data disjointedness and attribute overlap

Sixth, consider what a relationship between `SoftwareProject` and `HardwareProject` in Fig. 13 might look like, if **key equality, data overlap** and **attribute overlap** held on it. Because some attributes and data are common to both relations, this is an example of the "diamond-shaped" inheritance hierarchy, in which a class has generalization relationships to two superclasses and the two superclasses refer in turn to a common superclass.

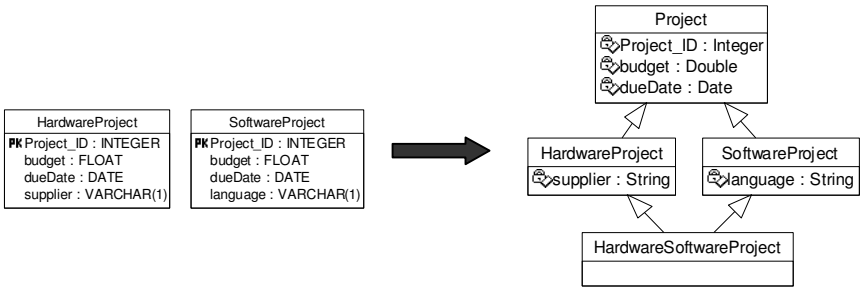


Figure 13. Mapping relationships: key equality, data overlap and attribute overlap

Finally, consider a relationship between `Employee` and `Project` in Fig. 14, when **key overlap** and **data equality** (or **data inclusion**) hold on it. Because all the information about departments in `Project` is also included in `Employee`, a closer examination of that relationship reveals that a relation `Department` is missing. Therefore, we "retrieve" `Department`, which has composite aggregations with both `Employee` and `Project`. That is, `Employee` and `Project` are indirectly related to each other through `Department`. (E.g. an employee works on a project controlled by a department he or she belongs to.)

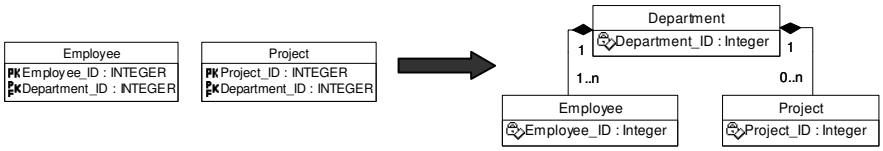


Figure 14. Mapping relationships: key overlap and data equality (or data inclusion)

4.5. Establishing cardinalities

Cardinality (or multiplicity) is the number of data instances that can participate in a relationship. To establish cardinalities, we need to consider foreign key values, as it is the ability of foreign key to be null and unique that determines if the relationship is zero-or-one-to-one, one-to-one, one-to-many or many-to-many, as shown in Table 2.

Table 2. Cardinalities for a foreign key in r2, which refers to r1

Foreign key values	r_1	r_2
Nullable and unique	0..1	0..1 or 1
Nullable and not unique	0..1	0..* or 1..*
Not nullable and not unique	1	0..* or 1..*
Not nullable and unique	1	0..1 or 1

In Fig. 15, assuming that in `Employee`, a foreign key attribute `Department_ID` cannot be null, we can see that every instance of `Employee` is associated with exactly one instance of `Department`. In addition, assuming that `Department_ID` is not unique, we can see that more than one instance of `Employee` can be associated with each instance of `Department`. Therefore, we establish a one-to-many relationship between `Department` and `Employee` (or many-to-one relationship between `Employee` and `Department`).

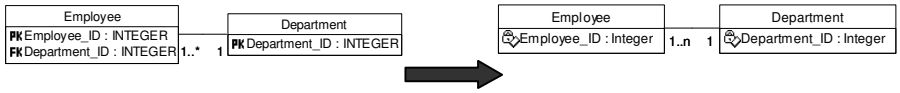


Figure 15. Establishing cardinalities

4.6. Alternatives

The schema transformation defines how a construct in the relational database schema is mapped to a (semantically equivalent) construct in the object database schema. However, generally, there can be several alternatives to this mapping. Some examples below show these alternatives more clearly.

In Fig. 16, `Department` has a cardinality of 0..1. Alternatively, we can introduce a subclass of `Employee`, say `Manager`, to represent those employees that manage departments. In some situations, however, the user may not be willing to introduce a new subclass. Therefore, we represent the relationship between `Department` and `Employee` as a zero-or-one-to-one association and express the constraint – that an employee can manage at most one department – with the cardinality alone.

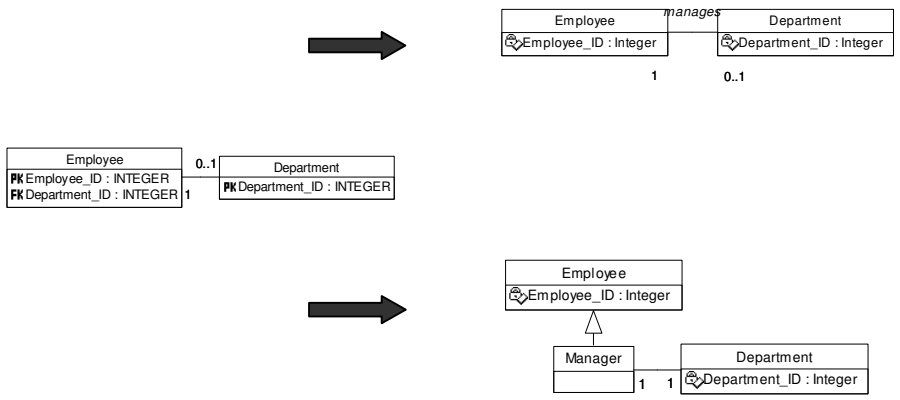


Figure 16. Mapping relationships: key disjoint and data equality. We can recast a zero-or-one-to-one association to a combination of inheritance and one-to-one association

Fig. 17 shows an association relationship between `Project` and `Task`. Alternatively, that relationship can be modeled as an aggregation, because logically projects are at a higher level than tasks. In particular, projects consist of tasks.

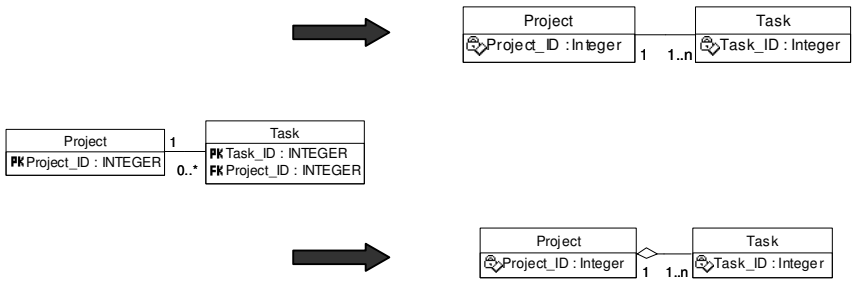


Figure 17. Mapping relationships: key disjoint and data inclusion. We can recast an association to an aggregation

5. Future work

A relationship between base (or dependent) relations is the most common kind of relationships. However, a relationship between composite relations gives rise to some special construct – a *constrained association* – that we need to consider, when transforming the relational database schema into the object database schema.

Constrained associations may arise in many situations, such as:

- *When two composite relations are connected by a time sequence*. For example, consider a relationship between `Works_on` and `Works_for` in Fig. 18, when **key overlap** and **data inclusion** hold on it. The former specifies that an employee works on a project, while the latter specifies that an employee works for a department. Of course, an employee cannot work on a project, if he or she has not worked for a department. Indeed, the employee is first employed by the department, and then he or she is assigned to the project. Therefore, the relationship between `Works_on` and `Works_for` maps to a constrained association. In particular, `Works_on` is a subset of `Works_for`

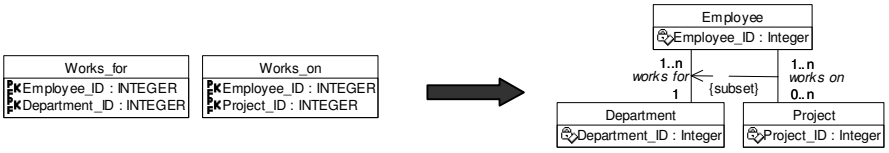


Figure 18. Mapping relationships: key overlap and data inclusion

- *When one composite relation represents a possibility, while another is the actual realization*. Indeed, we often use a constrained association to represent the fact that one entity can "do" something with another. For example,

consider a relationship between `Can_be_assigned_to` and `Works_on` in Fig. 19, when **key equality** and **data inclusion** hold on it. The former shows the mere possibility of instances of `Employee` being associated with instances of `Project`; i.e. an employee can be assigned to a project. But it does not depict the actual realization of this association, which is represented by a separate relation, `Works_on`, which captures what employees have been worked on projects[8]. Again, the relationship between `Can_be_assigned_to` and `Works_on` maps to a constrained association.

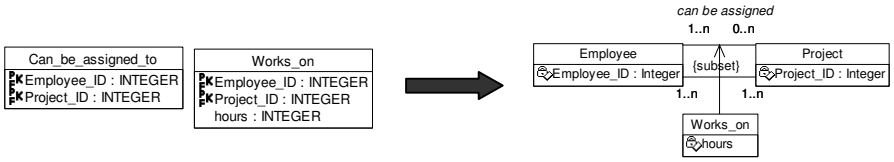


Figure 19. Mapping relationships: key equality and data inclusion

6. Conclusions

We have proposed a novel approach to reverse engineering of relational databases to object databases. Based on an analysis of key, data and attribute correlations, as well as their combination, our approach has two important advantages over the existing approaches in that:

- It allows the user to extract more semantics from relational databases, which include inheritance and optimization structures, and
- It reduces user interaction, which is mainly required to confirm the extracted semantics and to give them more appropriate names[9].

Acknowledgements

This research is partly sponsored by Estonian Science Foundation under the grant nr. 4067.

[8] The actual realization may involve additional attributes that are not part of the possibility. For example, in Fig. 19 `Works_on` has an additional attribute `hours`.

[9] The schema transformation process cannot be completely automated. User interaction is still necessary, when ambiguities occur and semantics cannot be inferred [4].

References

[1] J. Hainaut, J. Henrard, J. Hick, D. Roland and V. Englebert; Database Design Recovery, Proceedings of the 8th Conference on Advanced Information Systems Engineering, 1996, pp. 272 – 300

[2] J. Petit, F. Toumani, J. Boulicaut and J. Koulomdjian; Towards the Reverse Engineering of Denormalized Relational Databases, Proceedings of the 12th International Conference on Data Engineering, 1996, pp. 218 – 227

[3] W. Premerlani and M. Blaha; An Approach for Reverse Engineering of Relational Databases, Communications of the ACM, Vol. 37. No. 5, 1994, pp. 42 – 49

[4] A. Behm, A. Geppert and K. Dittrich; On the Migration of Relational Database Schemas and Data to Object-oriented Database Systems, Proceedings of the 5th International Conference on Re-technologies for Information Systems, 1997, pp. 13 – 33

[5] R. Chiang, T. Barron and V. Storey; A Framework for the Design and Evaluation of the Reverse Engineering Methods for Relational Databases, Data and Knowledge Engineering, Vol. 21. No. 1, 1996, pp. 57 – 77

[6] J. Albert, R. Ahmed, M. Ketabchi and M. Shan; Automatic Importation of Relational Database Schema in Pegasus, Proceedings of the 3rd International Workshop on Research Issues on Data Engineering: Interoperability in Multimedia Systems, 1993

[7] S. Ramanathan and J. Hodges; Reverse Engineering Relational Schemas to Object-oriented Schemas, Technical Report No. MSU-960701, Mississippi State University, 1996

[8] Z. Tari and J. Stokes; Designing the Reengineering Service for the DOK Federated Database System, Proceedings of the IEEE International Conference on Data Engineering, 1997, pp. 465 – 475

[9] Rational Rose; Using Data Modeler, version 1.3, 2001, http://www.rational.com

[10] Rational Rose; Unified Modeling Language (UML), version 1.3, 2001, http://www.rational.com

[11] D. Maier and S. Zdonik; Fundamentals of Object-oriented Databases, Readings in Object-oriented Databases, eds. D. Maier and S. Zdonik, 1990, pp. 1 – 32

SCIENTIFIC PAPERS UNIVERSITY OF LATVIA. 2004. VOLUME 672, Datbases and Information Systems
401–412 P.

MDA: Correctness of Model Transformations – The Level M0 Phenomenon

Karlis Podnieks

University of Latvia
Institute of Mathematics and Computer Science
29 Raina boulevard, Riga, LV-1459, Latvia
www.ltn.lv/~podnieks

Abstract. The paper considers the model transformations that are solving the working problem posed to the MOF QVT submitters, the so-called UML to RDBMS transformation. How to determine, is a proposed transformation correct, or not? The most natural solution: extend the model transformation (level M1) by including the corresponding uniform instance data transformation (level M0). If the latter one is "correct", then so is the former. Here we have the "level M0 phenomenon": it seems, in many cases, we cannot specify the correctness of model transformations, if we do not include correct instance data transformations. The problem: is this phenomenon inevitable? Couldn't we escape it – at least theoretically?

Keywords. Model transformations, MDA, model mappings, correctness, completeness.

1. The UML to RDBMS transformation

The Object Management Group (OMG) has issued a Request for Proposal for a Query/Views/Transformations (QVT) language that would allow defining of mappings between different information models [9] (OMG, 2002).

In [5] (Gerber et al, 2002), the authors ask: "In defining mappings from model to model, the question of correctness of the mapping arises. ... The more complex form of correctness is that of semantic correctness; does the result of transformation *mean* the same thing as the input?"

Indeed, let us consider a small fragment (see Figure 1) of the working problem posed to the MOF QVT submitters, the so-called UML to RDBMS transformation [9] (OMG, 2002).

The transformation problem is expressed as follows.

The input model is an interpretation of the input meta-model. It consists of persistent and transient classes owning attributes. Attributes may be primitive (having a

primitive data type), or complex (having a transient class as a type). The output model is an interpretation of the output meta-model. It consists of tables owning columns.

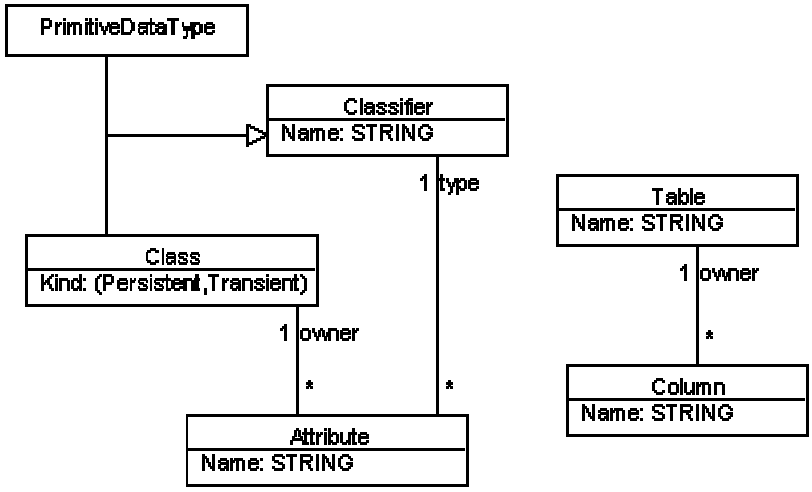


Figure 1. Fragments of UML and RDBMS meta-models

The transformation in question must: a) Transform each persistent class into a single table. b) Transform each class attribute of a primitive type into a column of the corresponding table. c) "Drill down" class attributes of complex types to leaf-level primitive attributes; transform these primitive attributes into columns of the corresponding table.

How could we determine, is a proposed transformation of this kind "correct", or not?

2. Two transformations of the same input model

Figure 2 represents an example input model – an interpretation of the fragment-UML meta-model from Figure 1.

2.1. An "absolutely complete" transformation

As a trivial example, the following transformation T1 should be regarded as "absolutely complete" (see also Figure 3):

a) T1 transforms a persistent class named C1 into a table named t_C1.

b) If the class C1 owns an attribute A1, and the type of A1 is the class C2, and C2 owns an attribute B1 of a primitive type STRING, then T1 transforms A2 into a column named c_A1_C2_B1_STRING.

c) Similarly, in all the other situations.

T1 is "absolutely complete", because it is completely reversible – **no information gets lost during the transformation**. Indeed, having an output model, generated by T1, we can restore the entire input model:

a) From the table named t_C1 - restore a persistent class C1.

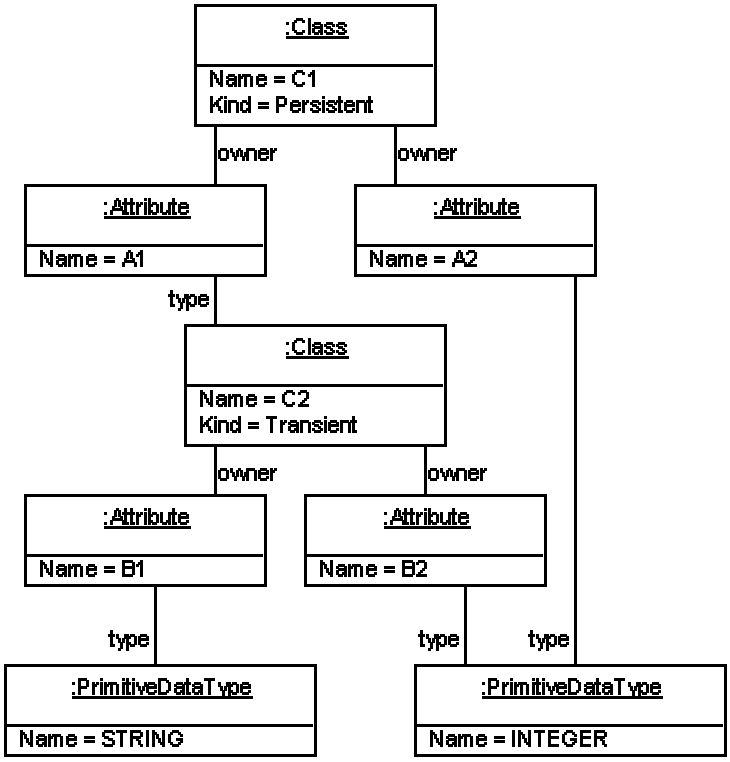


Figure 2. Example input model – an interpretation of the UML meta-model

b) From the column named c_A1_C2_B1_STRING, owned by the table t_C1 - restore: an attribute A1 owned by the class C1, the class C2 (if not restored earlier) - as the type of A1, an attribute B1 owned by C2, and the primitive type STRING (if not restored earlier) - as the type of B1.

c) Similarly, in all the other situations.

Thus, all elements of the input model can be restored from the output model.

Note. Of course - with the exception of transient classes that are not used as attribute (or sub-attribute) types in persistent classes. This small problem can be solved (as in [3] (Bernstein, 2003)) by requiring each model to have a root object, to which all the other objects must be connected via "is part of" relationships.

2.2. Practical transformations do not need to be "absolutely complete"

But, of course, none of the actual MOF QVT proposals includes the "absolutely complete" transformation T1 as its part (see, for example, [11] (QVT Partners, 2003) and [12] (Willink, 2003)). Instead of T1, they include another transformation T2, which differs from T1 as follows (see also Figure 4):

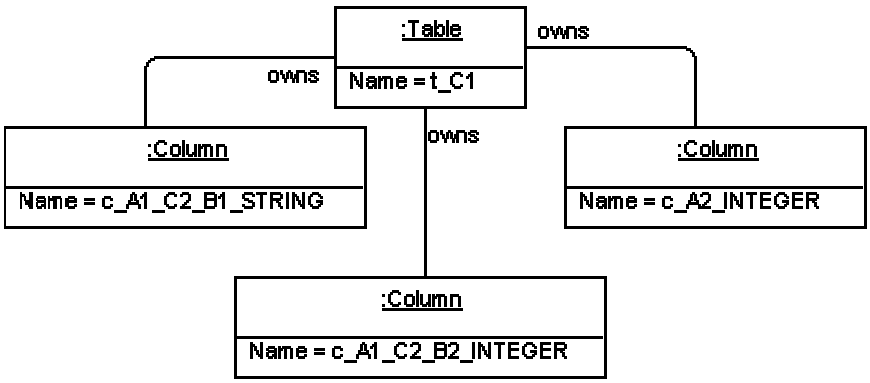


Figure 3. Example output model created by the transformation T1

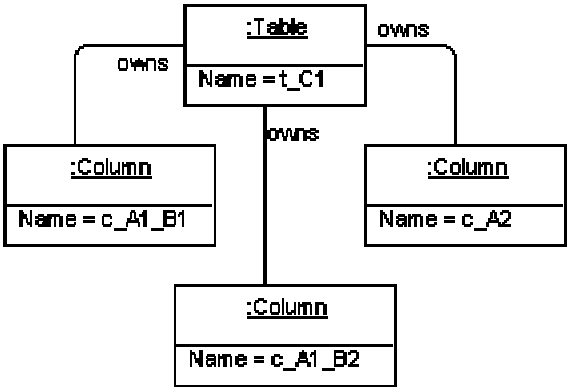


Figure 4. Example output model created by the transformation T2

b) If the class C1 owns an attribute A1, and the type of A1 is the class C2, and C2 owns an attribute A2 of a primitive type STRING, then T2 transforms C2 into a column named c_A1_A2.

When compared to the above T1's version of the column name (c_A1_C2_B1_STRING), T2, in its version c_A1_B1, omits the intermediate class name C2, and the primitive type name STRING. Thus, T2 is not completely reversible - the information about names of transient classes and primitive types **gets lost during the transformation**. Then, why should we regard this widely used T2 as a "correct" transformation? In which sense, the result of T2 "*means* the same thing as the input" [5] (Gerber et al, 2002)?

Although, perhaps, never spoken out explicitly, the intended semantics of the UML to RDBMS transformation is as follows. We do not need this transformation by itself. We need it as a **basis for instance data (database contents) transformations**. Indeed, the input UML model (level M1) can be regarded as a schema of an advanced object-oriented database (level M0), and the output RDBMS model – as a schema of a traditional relational database. Thus, in fact, to solve the UML to RDBMS transformation problem completely, we must provide not only the model (database schema) transformation. To make the model transformation useful, we must provide also the instance data (database contents) transformation that would allow converting (without loss of information) the contents of an advanced object-oriented database into the contents of a traditional relational database. Of course, the solution of this problem is a well-known topic described in the database textbooks for students.

And, of course, for the above small fragment of the problem (Figure 1), the database contents transformation D2 corresponding to the schema transformation T2 is trivial:

a) For an instance of a persistent class C1, create a row in the table t_C1.

b) Scan all instances of attributes of an instance of C1. If we meet an instance of a complex attribute A1, and the type of A1 is the class C2, then scan all instances of attributes of A1. If we meet an instance of a primitive attribute B1, and the value contained in the instance is "123", then, in the corresponding row of the table t_C1, create a cell corresponding to the column c_A1_B1, and containing the value "123".

c) Similarly, in all the other situations.

Thus, in the resulting database created by D2, the instances of intermediate complex attributes (like as A1 in the above example), and links connected to them, are completely ignored. But, nevertheless, D2 **is completely reversible**. Indeed, we can restore easily the contents of the input (object-oriented) database from the contents of the output (relational) database, if we can use, additionally, the information contained in the **(input and output) database schemas**:

a) For each row of the table t_C1, create an instance of the class C1.

b) For a cell corresponding to the column c_A1_B1, and containing the value "123", create (if not created before) an instance of the attribute A1, and link it to the

corresponding *owner* instance of C1, and create an instance of the attribute B1 containing the value "123", and link it to the corresponding *owner* instance of A1.

c) Similarly, in all the other situations.

Thus, by referring to database schemas, we can restore all the input database information, missing in the output database contents.

And thus, the pair T2+D2 can be regarded as a complete database schema transformation. And, of course, all the actual MOF QVT (and similar) proposals can be proved to be complete (for examples, see [11] (QVT Partners, 2003) and [12] (Willink, 2003)).

Note. Of course, in the MDA context, many transformations do not need to be complete even in the above-mentioned restricted sense. In MDA, transformations may lose information; they may merge parts of several models, add new information via user interfaces etc. In MDA, a model transformation is acceptable, if it performs its task.

3. The level M0 phenomenon

As we now see, it may happen that specifying the correctness (for example, the completeness) of model transformations becomes hard (if not impossible), if **we restrict the problem to the model level (level M1), and ignore model semantics (level M0).**

Perhaps, the simplest case when a general setting of this problem becomes possible by adding the level M0 considerations, are database schemas (see Figure 5). Here, a solution can be achieved, if we require extending of each database schema transformation T by a corresponding uniform data transformation D. If the data transformation D can be proved to be completely reversible, then T+D can be regarded as a complete schema transformation.

In fact, we have arrived here at the same "level M0 phenomenon" as several other researchers.

For example, in [6] (Kalinichenko, 1997), the author defines a general notion of data model mappings (see Definition 5). This definition includes data type **state space mappings** (i.e. database contents transformations) as a component that **cannot be reduced** to the schema level components. And, in the example mapping (of ODMG'93 data model into SYNTHESIS data model), the author defines the necessary state space mapping and verifies its correctness.

In [1] (Alagic, Bernstein, 2002), by using the language of category theory, the authors propose a general definition of a schema transformation framework (see, Definition 7). This definition includes **database morphisms** (i.e. database contents

transformations) as a component that **cannot be reduced** to the model level components. And, in the example framework (transformations between object-oriented database schemas with constraints) the authors define the necessary database morphisms (see Definition 15) and prove their correctness (Theorem 2).

In [3] (Bernstein, 2003), after considering several model management operators, the author concludes (see Section 3.10): "The model management operators defined in Section 3 are purely syntactic. That is, they treat models and mappings as graph structures, not as schemas that are templates for instances… Still, in most applications, to be useful, models and mappings must ultimately be regarded as templates for instances. That is, they must have semantics. Thus, there is a semantic gap between model management and applications that needs to be filled."

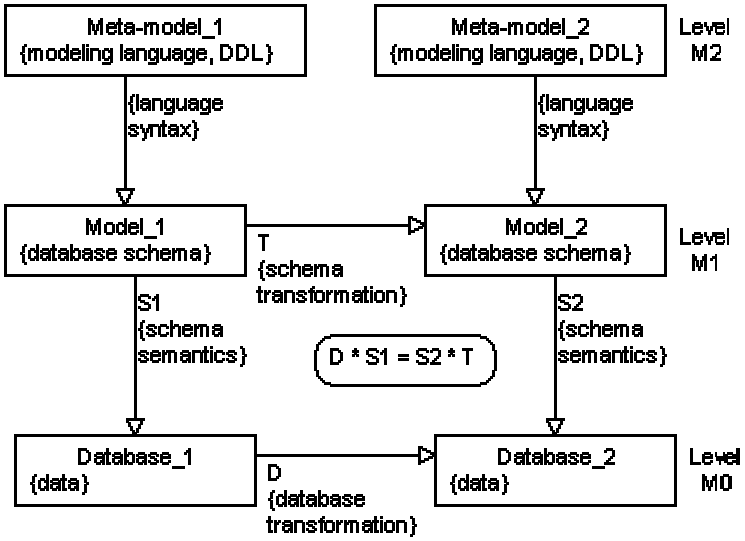


Figure 5. Database schema transformations

In [10] (Pottinger, Bernstein, 2003), extending [3] (Bernstein, 2003), the authors propose a generic version of the model management operator *Merge*. Among the directions of the future work they mention "showing that the *Merge* result, when applied to models and mappings that are templates for instances, has an appropriate interpretation on instances."

In [8] (McBrien, Poulovassilis, 1999), the authors propose an interesting class of automatically reversible transformations between database schemas. The reversibility is achieved by including database (i.e. instance data) queries into transformations. For example, the transformation *delNode(c, q)* (see Section 2.1) includes a query *q*, which should allow restoring the contents (i.e. the instances) of the entity class *c* from the database contents remaining after the deletion. Thus, here, the schema transformation framework includes level M0 considerations from the very beginning.

4. The nature of relationships between modeling layers M2-M1-M0

In [4] (Bezivin, Gerbe, 2001), the authors indicate that the level M0 may become significant even in software engineering. This argument can be generalized to cover any models having a kind of execution semantics, for example, UML state and activity diagrams (see Figure 6).

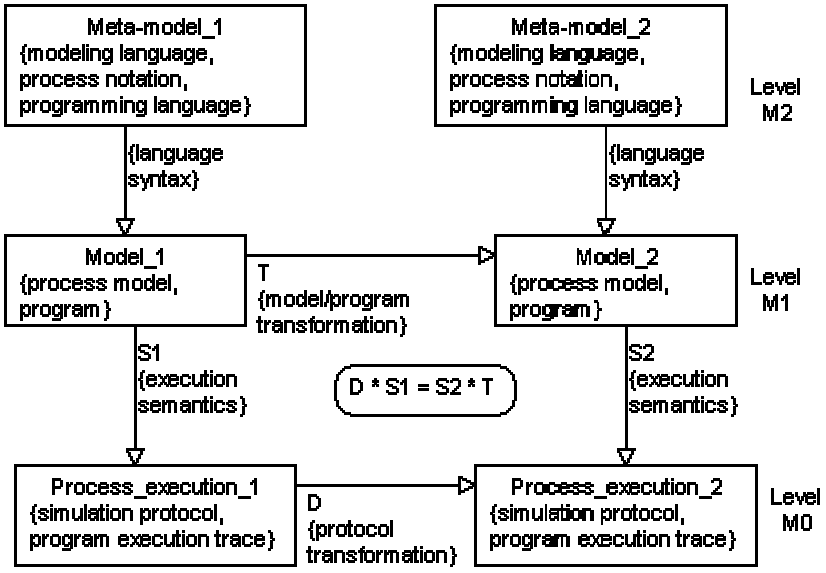


Figure 6. Process model transformations

In both of the above Figures 5 and 6, the relationship between the modeling layers M2 and M1 is indicated as "**language syntax**". Indeed, usually, a meta-model defines only the allowed syntax of the corresponding models, and not their semantics. If the meta-model is represented as a UML class diagram, then the "language syntax" is defined uniformly by applying the standard UML diagram semantics. Or, if the meta-model is represented by means of predicate logic, then the "language syntax" is defined uniformly by using the usual interpretations of (normally - many sorted) first order languages. But, as the result, at the level M1, we obtain models without semantics.

The relationship between the layers M1 and M0 should be called, on the contrary, "**model semantics**". Usually, this relationship is more complicated, and much more specific (i.e. less uniform) than the relationship between the layers M2 and M1. For example (see Figure 1), the RDBMS database schema semantics is essentially different from the UML schema semantics.

As an example, let us consider the "database schemas" represented in Figures 2, 3 and 4. In Figure 4, we see the "table" t_C1 (what's a table?), which owns three "columns" – c_A1_B1, c_A1_B2, and c_A2 (what's a column?). Of course, we know that a table is a collection of rows; each row consists of cells; each cell carries a value and corresponds to one of the columns; and, in each row, each column is represented by exactly one cell. But, of course, this knowledge cannot be derived from Figure 4, which represents only the data specific to the table t_C1, and not the general semantics of relational database tables.

However, this knowledge can be derived from an alternative RDBMS meta-model represented in Figure 7 (with the following constraint added: in each row of a particular table, each column – of this table - is represented by exactly one cell). Surprisingly, such a simple extension changes the nature of the RDBMS meta-model radically. Indeed, if, at the level M2, we will have the meta-model of Figure 7, then, at the level M1, we will have already… "two in one" – the database schema (i.e. table names with column names assigned), together with the database contents (i.e. cell values arranged in rows, columns and tables).

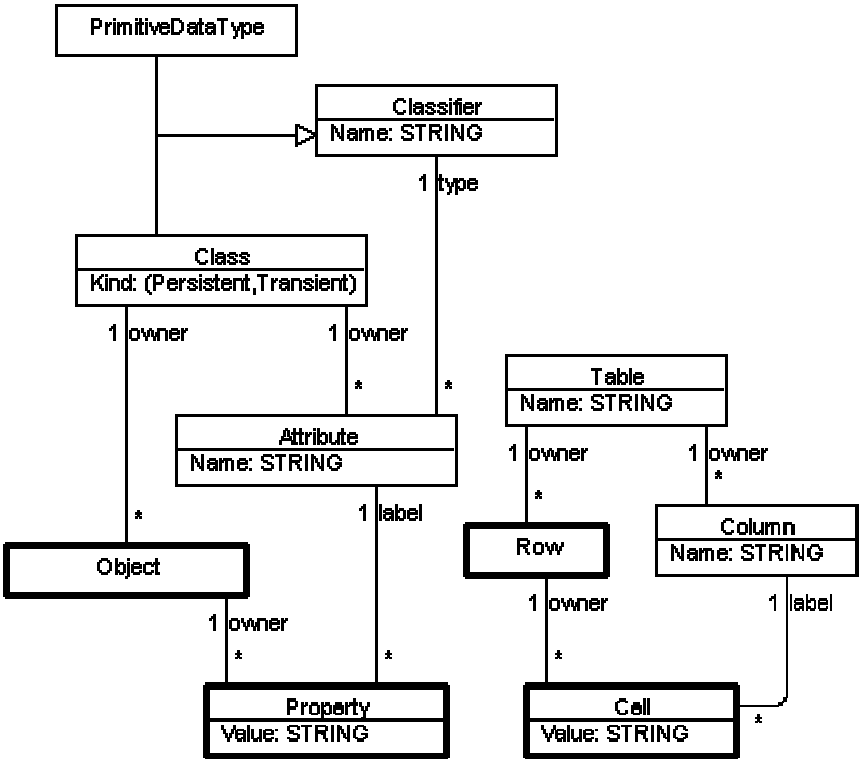


Figure 7. Fragments of alternative UML and RDBMS meta-models (compare with Figure 1).

To capture the intended semantics, the UML meta-model of Figure 1 can be extended in a similar way (see Figure 7, with the following constraint added: in each

object of a particular class, each attribute – of this class - is represented by exactly one property).

If OMG, in its Request for Proposal for QVT language [9] (OMG, 2002), would have used the Figure 7 style meta-models instead of the Figure 1 style ones, then the QVT partners would be forced to demonstrate that the proposed languages are good enough for simultaneous transformations of models and instance data (of course, they are, see [11] (QVT Partners, 2003) and [12] (Willink, 2003)).

5. The problem

Is the above-stated "level M0 phenomenon" inevitable? Couldn't we escape it – at least theoretically, and, at least, for some classes of model transformations?

Suppose, we have two different meta-models (level M2), and each of them defines the syntax of its own class of models at the level M1. Assume also, that we have precise definitions of semantics of these classes of models. Does this mean that, having these definitions only; we will be able, for each (complete, or correct, whatever it means) model transformation, to obtain automatically the corresponding correct uniform instance data transformation?

Could a correct uniform instance data transformation be derived from two model semantics definitions and a model transformation?

If, for some class of models and transformations, the answer would be positive, i.e., if there would be a general algorithm allowing to build automatically, for each model transformation (of this class), the corresponding (complete, or correct, whatever it means) instance data transformation, then the level M0 considerations would not add new information to the level M1, and transformation problems could be solved working exclusively at the level M1. Then, for this class of model transformations, the "level M0 phenomenon" would disappear…

But, if, for this class, such a general algorithm is impossible, then the model transformations (of this class) always must include, as an integral part, the corresponding instance data transformations. And, transformation specifications (for this class) must include the requirement of providing these level M0 transformations.

The database schema transformations proposed in [8] (McBrien, Poulovassilis, 1999) include database queries that allow building of the corresponding data instance transformations automatically.

6. Transformation specifications

Of course, **we should specify transformations before trying to develop them.** Defining and proving correctness of a proposed model transformation may be a non-trivial task (see above). Is here a general solution possible? How should we specify a transformation before we try to develop it? Without a **specification**, we will never be able to verify correctness of a proposed transformation. If there is an error in our transformation, how could we detect it without a specification?

In [2] (Appukuttan et al, 2003], the authors propose to specify transformations by using **relations** (i.e., in general case, non-executable, "multi-directional" transformations, see Section 4.1). The next step – transformation implementations should be **mappings** (i.e. operational, "potentially uni-directional" transformations). How this approach is working in practice – see [11] (QVT Partners, 2003).

In [7] (Madhavan et al, 2002), the authors propose "a powerful framework for defining languages for specifying mappings and their associated semantics" (see Abstract). Mapping specifications (see Definition 1) are represented here by sets of formulas, i.e. they are, in fact, a kind of relations.

In terms of the above three-level diagrams (Figures 5 and 6), these specifications have the form P(S1, S2), where S1, S2 are interpretations, and P is a complicated relation. They specify, "which pairs of interpretations can co-exist, given the mapping" (see Section 3).

Thus, if P would be a "maximally strong" relation, then, for each S1, only one S2 would satisfy P(S1, S2), and, theoretically, we could hope to "compute" S2 from S1, thus, solving the "equation" D*S1=S2*T with respect to D. Indeed, the task of the transformation D is, in fact, converting of any S1 into the corresponding S2 (the model transformation T helping the process).

For which classes of models are such "maximally strong" relations P feasible? This question brings us back to the above-stated problem. Indeed, by "computing" D from P, we could escape the "level M0 phenomenon"…

Acknowledgements

I'm grateful for valuable discussions and comments provided by my colleagues, especially Janis Barzdins, Audris Kalnins and Martins Opmanis. Science Council of Latvia has funded the work reported in this paper under project Nr.02 0002.

References

[1] S. Alagic, Ph. A. Bernstein. A model theory for generic schema management. Proceedings of International Workshop on Database Programming Languages (DBPL '01), Lecture Notes in Computer Science 2397, 2002.

[2] B. Appukuttan, T. Clark, S. Reddy, L. Tratt, R. Venkatesh. A Model Driven Approach to Building Implementable Model Transformations. Workshop in Software Model Engineering (WiSME@UML'2003), October 21, 2003, San Francisco, USA.

[3] Ph. A. Bernstein. Applying model management to classical meta data problems. Proceedings of the Conf. on Innovative Database Research (CIDR), 2003, pp. 209-220

[4] J. Bezivin, O. Gerbe. Towards a Precise Definition of the OMG/MDA Framework. ASE'01, Automated Software Engineering, San Diego, USA, November 26-29, 2001.

[5] A. Gerber, M. Lawley, K. Raymond, J. Steel and A. Wood. Transformation: The Missing Link of MDA. Proceedings of Graph Transformation: First International Conference (ICGT 2002), October 7-12, 2002, Barcelona, Spain, Lecture Notes in Computer Science, vol. 2505, Springer-Verlag, 2002, pp. 90-105.

[6] L. A. Kalinichenko. Method for Data Models Integration in the Common Paradigm. Proceedings of the First East European Symposium on "Advances in Databases and Information Systems", St. Petersburg, September 1997.

[7] J. Madhavan, Ph. A. Bernstein, P. Domingos, A. Y. Halevy. Representing and Reasoning about Mappings between Domain Models. Proceedings of 18th National Conference on Artificial Intelligence (AAAI'2002), Edmonton, Canada, 2002.

[8] P. McBrien, A. Poulovassilis. Automatic migration and wrapping of database applications – a schema transformation approach. Proceedings of the 18th International Conference on Conceptual Modeling / the Entity Relationship Approach, Paris, France, November, 15-18, 1999, Springer Verlag, LNCS 1728, pp. 96-113

[9] OMG, "Request for Proposal: MOF 2.0/QVT", OMG Document, ad/2002-04-10. Available at http://www.omg.org/cgi-bin/doc?ad/2002-04-10

[10] R. A. Pottinger, Ph. A. Bernstein. Merging Models Based on Given Correspondences. Proceedings of the 29th VLDB Conference, Berlin, Germany, 2003.

[11] QVT Partners. Initial submission for MOF 2.0 Query / Views / Transformations RFP. Version 1.0 (2003.03.03). Available at www.qvtp.org/downloads/1.0/qvtpartners1.0.pdf

[12] E. D. Willink. A concrete UML-based graphical transformation syntax – The UML to RDBMS example in UMLX. Proceedings of Workshop on Metamodelling for MDA, University of York, England, 24-25 November 2003.

SCIENTIFIC PAPERS UNIVERSITY OF LATVIA. 2004. VOLUME 672, Datbases and Information Systems
413–422 P.

MDA as a Telecommunications Network Documentation Tool

Guntis Barzdins

University of Latvia, IMCS
29 Raina boulevard, Riga, Latvia
guntis@latnet.lv

Abstract. Although UML language and MDA (Model Driven Architecture) approach primarily are being developed for the needs of large software projects, these powerful ideas are fruitfully applicable also towards smaller projects, where having a precise model of a real-life system is essential. This case study illustrates how a complex telecommunications network can be precisely documented by means of an "executable model". Such executable model takes the form of a highly integrated spreadsheet, which is semi-automatically constructed from the OCL constrained UML description of the network structure. We also stipulate that this approach can be extended into generic MDA tool for creating complex spreadsheet applications from their OCL constrained UML models.

Keywords. Network documentation, MDA, spreadsheet programming

1. Introduction

Telecommunication systems and services are a notoriously complex domain consisting of a large number of subsystems. Industry-wide standards for dealing with this complexity in standardized manner are emerging, like eTOM (Enhanced Telecom Operations Map) [1], which provide a top-to-bottom view on the involved systems and interfaces.

Meanwhile the logical distance between such top-to-bottom view and the actual telecommunications transmission and switching network is so great, that in many cases an independent bottom-to-top approach is still appropriate. In this paper we focus on the bottom-to-top part of telecommunications operation IT support, which is heavily dependent on physical transmission technologies and architectures involved, and thus requires grassroots design of optimal IT support structure in each individual case. We claim in this paper that UML based MDA (Model Driven Architecture) is well suited for this kind of applications, and illustrate this by a case study from satellite communications domain.

On the methodological side, we claim that UML modeling and spreadsheet programming are a convenient and little explored technology match for implementing small, but logically advanced documentation projects.

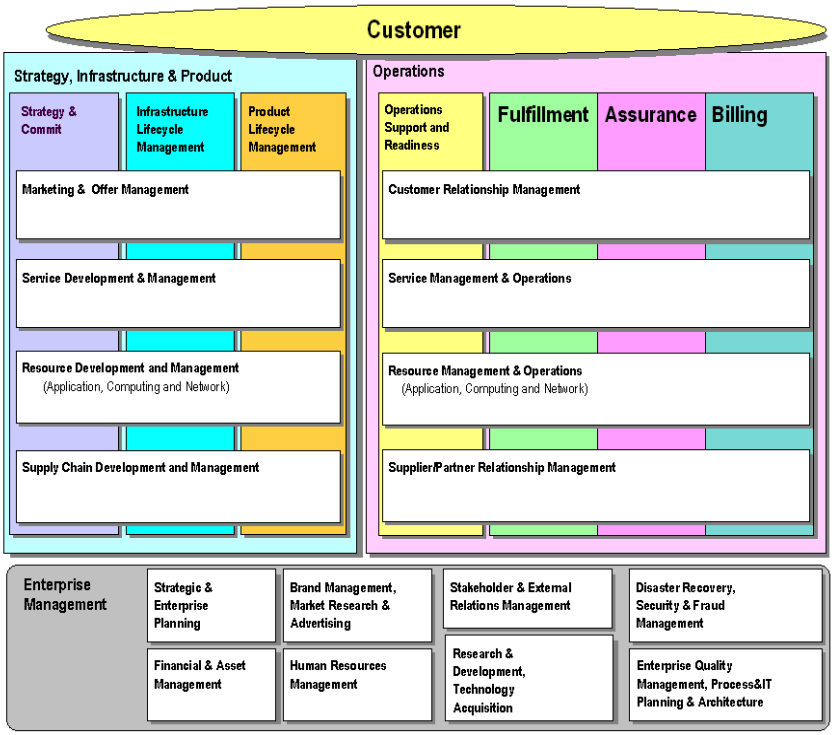


Figure 1. eTOM: enhanced Telecom Operations Map.

2. MDA tools used in this case study

Although theoretically any MDA tool could be used for such bottom-to-top cases, our observation is that the relatively small scale of these projects requires techniques optimized for quick (but not necessarily very efficient) implementations. We have found that UML modeling [2] and spreadsheet programming [3] provide a perfect match for these cases – powerful design techniques and simple implementation tools.

Despite their seemingly worlds-apart uses, the UML data modeling and spreadsheet programming have a common background – they both originate from the entity-relationship data representation model. It is obvious for UML class diagrams, but is true also for spreadsheet programming (although casual MS Excel spreadsheet user might not be aware of it). The full power of spreadsheet programming comes from lookup functions (LOOKUP, MATCH, INDEX, COUNTIF, SUMIF, etc.), which can be applied between worksheets (independent tables) of the workbook (database). We illustrate that lookup functions available in MS Excel cover the essential features of OCL – the Object Constraint Language [4], which is part of UML specification and in its expressive power is comparable to SQL for RDBMS. Therefore entity-relationship data representation model within

this paper is used as a "glue" between more abstract UML modeling and more pragmatic spreadsheet programming. The effect of such integration is three-fold:

- It provides means for designing spreadsheets of unmatched complexity, which adequately capture the data structure of the real-world system to be modeled.
- It yields a clear UML-style documentation of the spreadsheet itself. UML model of the spreadsheet (which at the same time is also the UML model of the real-world system to be modeled) gives a new way of communicating spreadsheet internal logic between its users.
- Due to its clear structure, spreadsheet programming provides a simple transformation path from OCL constrained UML model to its executable implementation. This, besides being compliant with MDA (Model Driven Architecture) framework [5], also suggests a useful extension to traditional spreadsheet software, like MS Excel, where spreadsheet application could be semi-automatically constructed from its OCL constrained UML description.

The rest of the paper is organized as follows: first we give a brief description of the real-world telecommunications system documentation problem, which triggered development of the described approach. Then we show in detail how this problem was solved using UML modeling and spreadsheet programming techniques, leading to an easy maintainable documentation (model) of a complex and frequently changing telecommunications system. The final part of the paper discusses the techniques used throughout the case study, and how they potentially could be extended into a more generic and automated tool.

3. Telecommunications system documentation problem

For telecommunications system engineers it is common to use diagrams like shown in Figure 2 to depict the structure of the particular telecommunications network setup. Such diagrams are sometimes accompanied with ad-hoc support documents listing telephone numbers, IP addresses, frequencies etc. configured on various elements of the telecommunications network. Only for very large and homogenous telecommunications networks (like public telephone network or large and homogenous ISP network) it is feasible to develop a specialized database application, where all essential configuration details are stored. For smaller and less homogenous networks it is usually left up to the telecommunications engineer to come up with the appropriate documentation, suitable for maintenance of the communications system throughout its lifetime. The result in this case is highly dependent on the presentation skills of the engineer.

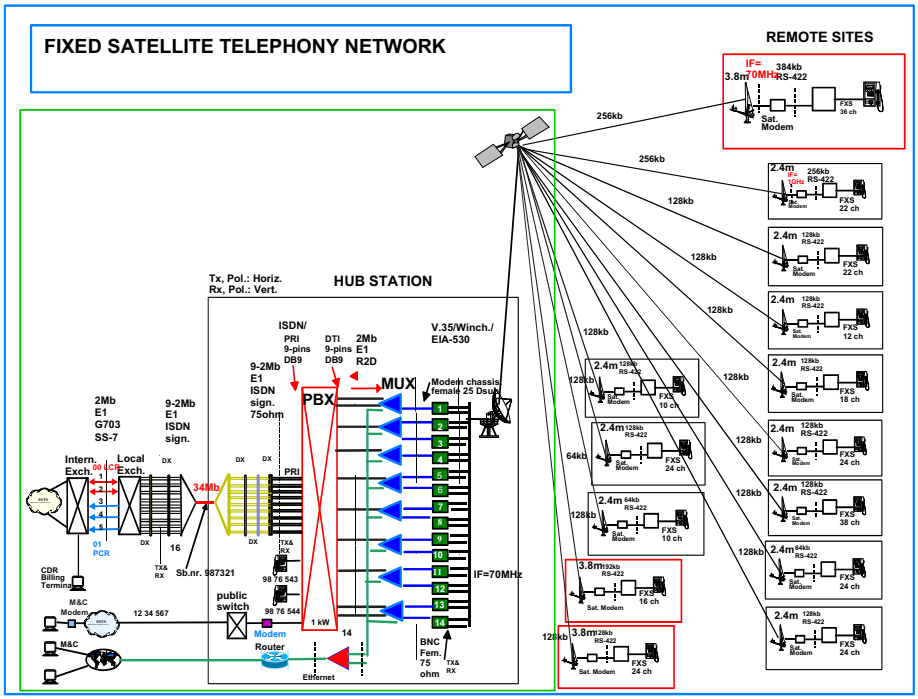


Figure 2. Traditional diagram of the satellite telephony network.

But the situation with such diagrams and ad-hoc support documents easily gets out of hand, if the system is subject to frequent configuration changes – usually proper documentation of changes requires massive modifications at various parts of the documentation. And there are no any safeguards against human-error – like changes inconsistently reflected in various parts of the documentation. Result of this phenomenon is that complex and frequently changing telecommunications systems are extremely hard to maintain error-free.

Therefore a desired documentation method must avoid redundancies (so that the same network element does not need to be repeatedly described in several parts of the documentation – a common source of documentation errors), and must provide safeguards against human-error (for example, automatic crosschecks, which would reveal the logical inconsistencies within the documentation). Such semi-automatic documentation system would also make configuration changes virtually error-free – the documentation of the intended new configuration can be crosschecked for consistency even before it is implemented, thus avoiding system downtimes due to ill-planned changes.

A proper MS Excel spreadsheet seemed to be a natural solution to this documentation problem (development and ongoing support of the specialized software would be an overkill). But how such highly complicated spreadsheet can be designed? This is where we decided to turn to UML design methodology. Afterwards it came bit of a surprise, how well the two technologies actually complement each other.

4. System description phase

From object-modeling point of view, the system diagram depicted in the Figure 2 is effectively an instance diagram of a yet to-be-drawn object model of this system. Analysis of the system revealed the functionally essential elements, shown in the form of UML class diagram in Figure 3. This model looks very different from the drawing in Figure 2, because it is concerned primarily with the functionally essential aspects of the system, and not with the physical layout of the components. This transition is natural from the system-analyst point of view, but might seem quite unnatural from the communications engineer point of view.

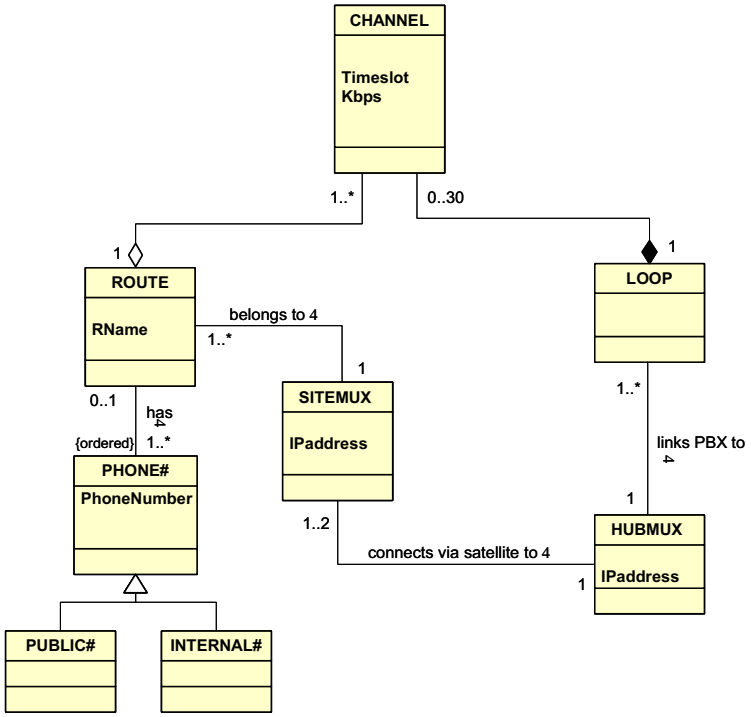


Figure 3. UML object model (class diagram) of the satellite telephony system.

Following comments should help to understand the object-model depicted in Figure 3:

- ⋕ CHANNEL represents a single telephone line. It has two main attributes – compressed voice bandwidth in Kbps (when transmitted via satellite) and timeslot (when transmitted via LOOP).
- ⋕ LOOP represents a bundle of up to 30 voice CHANNELs, which are physically transported over the same cable between the PBX (telephone switch) and HUBMUX (voice compression equipment).
- ⋕ ROUTE represents a group of voice channels, which have the same telephone numbers associated in the PBX. Note that channels that belong to the same route can be scattered over several LOOPs.

- PHONE# represents the phone number assigned to the route. Several phone numbers may be assigned to the same route.
- There are two types of telephone numbers: PUBLIC numbers, which can be dialed from public network, and INTERNAL numbers, which can be reached only within the PBX.
- HUBMUX is the voice compression equipment used at the hub side of the network. It can serve several remote SITEMUXes.
- SITEMUX is the voice compression equipment used at the remote location connected via satellite link. Note that voice CHANNELs belonging to the same ROUTE can span several LOOPs, but they all must end up in the same SITEMUX.

The next step is to transform this purely conceptual UML class diagram into the spreadsheet. As the target environment we will use standard MS Excel spreadsheet program.

The transformation is based on viewing UML class diagram as an entity-relationship data model, and then creating the tables (MS Excel worksheets) with columns matching the attribute names of the classes. Figures 3 and 4 illustrate this transformation – first, additional attributes (primary keys, foreign keys, selectors) are added to the classes to uniquely encode class relationships (Figure 4). Then this modified class diagram is mechanically transformed into the Excel workbook: separate worksheets represent each class of the diagram, while columns in the worksheets represent attributes of the corresponding classes (Figure 5). Rows in the resulting worksheets represent individual class instances, with columns showing attribute values for each instance.

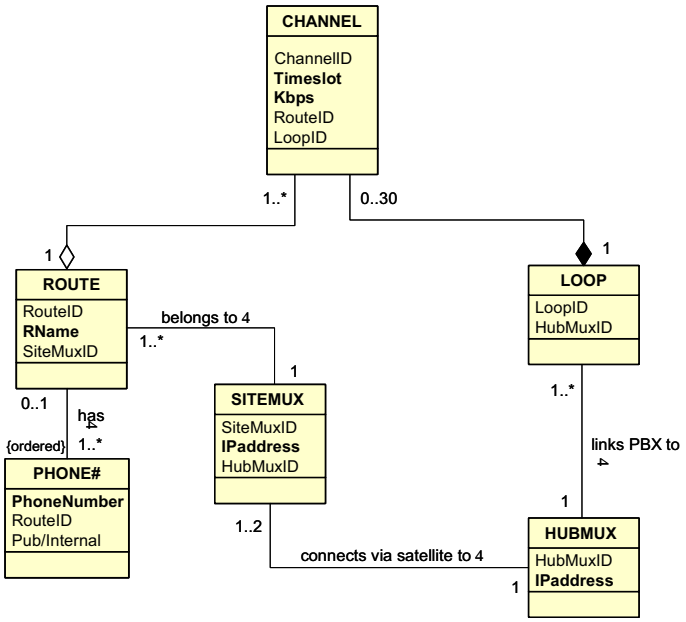


Figure 4. Additional attributes are added to encode relationships between the classes.

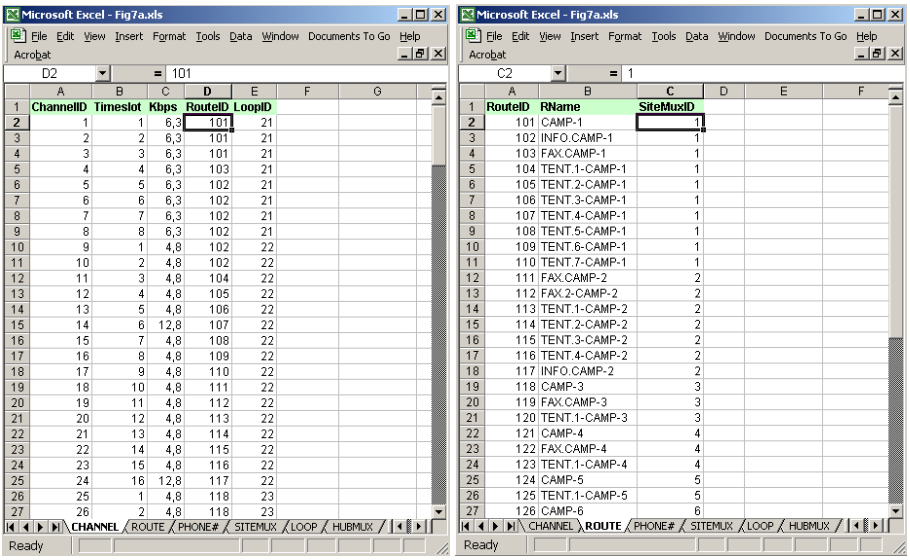


Figure 5. MS Excel workbook corresponding to the class diagram in Figure 4.

By filling the newly created MS Excel workbook with data, we can encode there the actual configuration of the real-world telecommunications system. Note that this MS Excel workbook contains full information about the telecommunications system configuration and, importantly, – it avoids any information redundancy (no configuration information is being duplicated anywhere in the documentation).

But this workbook still has two drawbacks, which we will need to fix in the next section:

- It is not easy to find information in this workbook. For example, to find an answer to the question *"To which site a particular channel belongs to?"*, one would need to search through several tables.
- There are no any safeguards against human-error.

5. Report generation and constraint checking phase

If the previous chapter was dominated by UML design techniques to reveal the functional structure of the system and to encode it into MS Excel workbook in the non-redundant manner, then now is the time to unleash the power of spreadsheet programming. The essence of this phase is following: out of the system description columns created in the previous chapter (which fully and without redundancy describe the system), we can use spreadsheet formulas to pre-calculate any useful attribute of the system.

A "useful attribute" of the system can be specified either in natural language, or by means of OCL – Object Constraint Language [4], an add-on part of UML. Below are listed some (not all) useful attributes, part of which are depicted also in Figure 6. For comparison, attributes are defined in three different syntaxes – natural language, OCL, and MS Excel:

∉# Add a column to Channels worksheet, where formula calculates the SiteID to which this channel is routed.

OCL notation: `Channel: self.Route.SiteMux.SiteMuxID`

Excel formula: `=INDEX(ROUTE!C:C;MATCH(D2;ROUTE!A:A;);1)`

∉# Add a column to Channels worksheet, where formula calculates the HubMuxID through which this channel is routed.

OCL notation: `Channel: self.Route.SiteMux.HubMux.HubMuxID`

Excel formula: `=INDEX(SITEMUX!C:C;MATCH(INDEX(ROUTE!C:C;MATCH(D2;ROUTE!A:A;);1);SITEMUX!A:A;);1)`

∉# Add a column to Routes worksheet, where formula calculates one of phone numbers assigned to this Route.

OCL notation: `Route: self.Phone#.PhoneNumber->first`

Excel formula: `=INDEX('PHONE#'!A:A;MATCH(A2;'PHONE#'!B:B;);1)`

Besides that, we can create a new worksheet containing some crosscheck and overview values of the system, for example:

∉# Total number of channels.

OCL notation: `Channel.allInstances->size`

Excel formula: `=COUNTIF(CHANNEL!A:A;">0")`

∉# Total number of phone numbers in use.

OCL notation: `Phone#.allInstances->select(RouteID>0)->size`

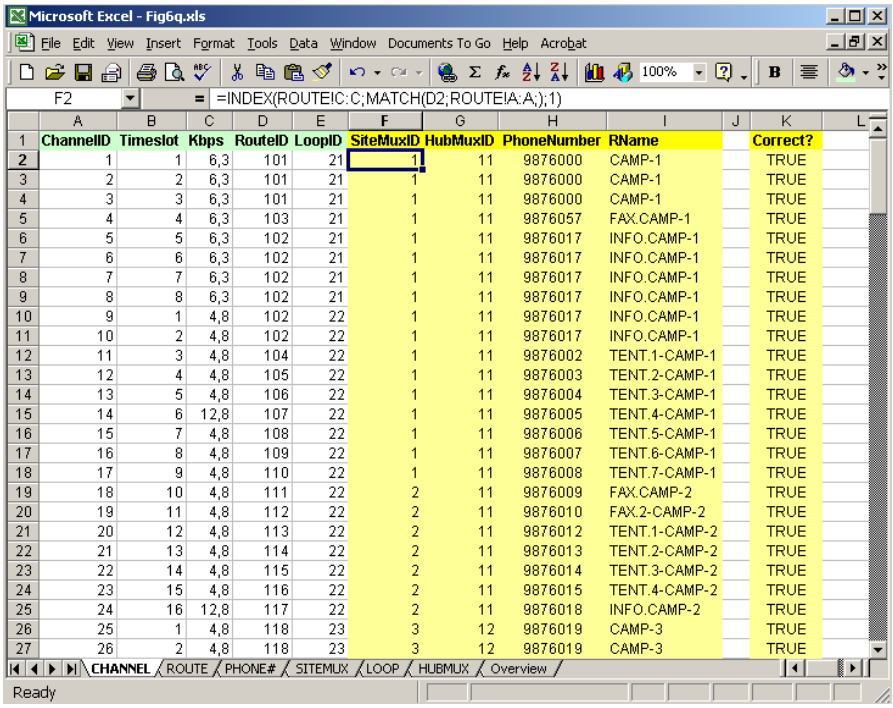
Excel formula: `=COUNTIF('PHONE#'!B:B;">0")`

∉# Total number of misrouted channels: the Loop, to which Channel C belongs, is connected to the HubMuxID=A, but the Route associated with channel C belongs to SiteMux, which is connected to HubMuxID=B, and A <>B.

OCL notation: `Channel.allInstances ->`
`select(Route.SiteMux.HubMuh <> Loop.HubMux)->size`

Excel formula: first mark misrouted channels in a separate column K on Channels worksheet by formula
`=(INDEX(LOOP!B:B;MATCH(E2;LOOP!A:A;);1)=INDEX(SITEMUX!C:C;MATCH(INDEX(ROUTE!C:C;MATCH(D2;ROUTE!A:A;);1);SITEMUX!A:A;);1))`
and then calculate `=COUNTIF(CHANNEL!K:K;FALSE)`

Such overview values are specifically oriented towards crosschecking the system description integrity – for example, if our intention was to change only telephone number assigned to some route, then it should not change the overview value for the number of phone numbers in use. And the number of misrouted channels, of course, must be 0 at all times.



	A	B	C	D	E	F	G	H	I	J	K	L
1	ChannelID	Timeslot	Kbps	RouteID	LoopID	SiteMuxID	HubMuxID	PhoneNumber	RName		Correct?	
2	1	1	6,3	101	21	1	11	9876000	CAMP-1		TRUE	
3	2	2	6,3	101	21	1	11	9876000	CAMP-1		TRUE	
4	3	3	6,3	101	21	1	11	9876000	CAMP-1		TRUE	
5	4	4	6,3	103	21	1	11	9876057	FAX.CAMP-1		TRUE	
6	5	5	6,3	102	21	1	11	9876017	INFO.CAMP-1		TRUE	
7	6	6	6,3	102	21	1	11	9876017	INFO.CAMP-1		TRUE	
8	7	7	6,3	102	21	1	11	9876017	INFO.CAMP-1		TRUE	
9	8	8	6,3	102	21	1	11	9876017	INFO.CAMP-1		TRUE	
10	9	1	4,8	102	22	1	11	9876017	INFO.CAMP-1		TRUE	
11	10	2	4,8	102	22	1	11	9876017	INFO.CAMP-1		TRUE	
12	11	3	4,8	104	22	1	11	9876002	TENT.1-CAMP-1		TRUE	
13	12	4	4,8	105	22	1	11	9876003	TENT.2-CAMP-1		TRUE	
14	13	5	4,8	106	22	1	11	9876004	TENT.3-CAMP-1		TRUE	
15	14	6	12,8	107	22	1	11	9876005	TENT.4-CAMP-1		TRUE	
16	15	7	4,8	108	22	1	11	9876006	TENT.5-CAMP-1		TRUE	
17	16	8	4,8	109	22	1	11	9876007	TENT.6-CAMP-1		TRUE	
18	17	9	4,8	110	22	1	11	9876008	TENT.7-CAMP-1		TRUE	
19	18	10	4,8	111	22	2	11	9876009	FAX.CAMP-2		TRUE	
20	19	11	4,8	112	22	2	11	9876010	FAX.2-CAMP-2		TRUE	
21	20	12	4,8	113	22	2	11	9876012	TENT.1-CAMP-2		TRUE	
22	21	13	4,8	114	22	2	11	9876013	TENT.2-CAMP-2		TRUE	
23	22	14	4,8	115	22	2	11	9876014	TENT.3-CAMP-2		TRUE	
24	23	15	4,8	116	22	2	11	9876015	TENT.4-CAMP-2		TRUE	
25	24	16	12,8	117	22	2	11	9876018	INFO.CAMP-2		TRUE	
26	25	1	4,8	118	23	3	12	9876019	CAMP-3		TRUE	
27	26	2	4,8	118	23	3	12	9876019	CAMP-3		TRUE	

Figure 6. MS Excel workbook from Figure 5, populated with additional calculated attributes.

MS Excel workbook shown in Figure 6 is the final result of our design efforts (full example is available at `http://www.ltn.lv/~guntis/mdaex.xls`). The resulting spreadsheet has following important properties:

- Now we can find any useful information about the system easy. For example, answer to the question *"To which site a particular channel belongs to?"* is automatically pre-calculated and visible in the spreadsheet line describing the channel.
- There is no information redundancy – each system parameter is stored (and modified) only in one place. From there, all calculated cells are updated with correct values automatically.
- Overview parameters of the system can be used to spot description inconsistencies.

Note that this spreadsheet is a true model of the real-world communications system itself – the telecommunications engineer can spend more time designing (and crosschecking) changes to the system in this virtual model, rather than in the physical system. Only when the changes seem OK in the model, engineer proceeds with their physical implementation. Thus documentation is always up-to-date – it is actually updated already in the change design phase, before the physical change is even implemented.

6. Generic MDA framework for spreadsheet applications

The key remaining question is – how universally applicable is the well-working approach illustrated in this case study? For reasons mentioned below, we suggest that it is generally applicable to tasks, where OCL constrained UML static structure model captures majority of system logic. Development of semi-automatic tools supporting this approach could potentially spark their rather massive use due to ubiquity of MS Excel spreadsheet users.

From data representation point of view spreadsheets are similar to relational databases – they both support lookup functions and can contain embedded procedures, where bulk of data-driven business logic can be embedded. In both cases external code should be used to implement interactivity (note, that also spreadsheet program can interact with external code through `SetCellValue(…)` and `GetCellValue(…)` methods – see details in [3] about server-based spreadsheet engines).

From MDA methodology point of view transformations used in the case study must be fairly easy to encode in a formal transformation description language, like the one proposed in [6]. In this case MS Excel workbook shown in Figure6 could be generated by automatic (or semi-automatic) transformation from UML class diagram in Figure 3 and OCL constraints defined in Section 5. This would result in a clean MDA framework for creating complex Excel spreadsheets from UML and OCL models. Although we are not aware of any tool being developed along these ideas, the implementation appears to be rather straightforward and would result in substantially enhanced complex spreadsheet development tool.

Our conclusion is that spreadsheet programming based implementation of UML models should not be neglected, as this might be adequate for smaller projects, fast prototyping, or learning purposes. The case study shows that it works quite well at least in telecommunications operation environment.

References

[1] TMFORUM: NGOSS and eTOM. http://www.tmforum.org/TMFC1859 eTOM Overview.pdf (2003)
[2] G. Booch, I. Jackobson, J. Rumbaugh. The Unified Modeling Language. Reference Manual, Addison-Wesley, 1999.
[3] M. Smialek. Spreadsheet Programming: MS Excel as Component Development Environment, http://www.devx.com/enterprise/Article/11686 (2003)
[4] J. Warmer, A. Kleppe. The Object Constraint Language: Precise Modeling with UML, Addison-Wesley, 1999
[5] OMG: MDA Guide Version 1.0.1, http://www.omg.org/docs/omg/03-06-01.pdf (2003)
[6] E. D. Willink. A concrete UML-based graphical transformation syntax – The UML to RDBMS example in UMLX. Workshop on Metamodeling for MDA, University of York, England, November 2003

ONTOLOGIES

SCIENTIFIC PAPERS UNIVERSITY OF LATVIA. 2004. VOLUME 672, Datbases and Information Systems
425–435 P.

Merging DAML+OIL Ontologies

Patrick Lambrix, He Tan

Department of Computer and Information Science
Linköpings universitet, Sweden
patla@ida.liu.se

Abstract. Ontologies are being used nowadays in many areas. Within each area there are a number of ontologies, each with their own focus, that contain overlapping information. In applications using multiple ontologies it is therefore of interest to be able to merge these ontologies. In this paper we describe a prototype implementation of SAMBO, an ontology merge tool for DAML+OIL ontologies. The tool generates suggestions for merging concepts and relations and for creating is-a relationships between concepts. We evaluate different strategies for the generation of suggestions. We also compare our tool with the ontology merge tools Protégé-2000 with PROMPT and Chimaera in terms of the quality of suggestions and the time it takes to merge ontologies using these tools.

Keyword. ontology merging, ontologies

1. Introduction

Intuitively, ontologies can be seen as defining the basic terms and relations of a domain of interest, as well as the rules for combining these terms and relations. Ontologies are being used nowadays in many areas for communication between people and organizations, as the basis for interoperability between systems, and as query models and indexes to repositories of information.

Within an area there are always a number of ontologies, each with their own focus. For instance, in bioinformatics (e.g. [8]), the ontologies cover different aspects in molecular biology such as molecular function and cell signaling. Many of these ontologies contain overlapping information. For instance, a protein can be involved in both cell signaling and other biological processes. In applications using ontologies it is therefore of interest to be able to use multiple ontologies. However, to obtain the best results, we need to know how the ontologies overlap and align them or merge them into a new ontology. Another reason for merging ontologies is that it allows for the creation of ontologies that can later be composed into larger ontologies. Also, companies may want to use de facto standard ontologies and merge them with company-specific ontologies.

In this paper we describe a prototype of the ontology merge tool SAMBO. In its current implementation it helps users merge DAML+OIL ontologies and performs description logic reasoning. We describe DAML+OIL in section 2 and SAMBO in section 4. Further, we describe a number of evaluations in section 5. SAMBO can generate suggestions for merging, but allows the user to choose different algorithms

for the generation of these suggestions. In the first evaluation we compare the different algorithms. In the second evaluation we compare SAMBO with two well-known ontology merge tools, Protégé-2000 with PROMPT and Chimaera (described in section 3), regarding the quality of the suggestions and the time it takes to merge two ontologies using these tools.

2. DAML+OIL

DAML+OIL [4] is an ontology language built on Web standards such as XML and RDF. It takes a frame-based approach, and inherits the expressiveness and reasoning power from description logics. DAML+OIL can be seen as a highly expressive description logic. More precisely, DAML+OIL is equivalent to the $\mathcal{SHIQ}$ description logic with the addition of existentially defined classes and data types [7]. This equivalence allows DAML+OIL to exploit implemented description logic systems to provide highly optimized reasoning services. DAML+OIL has already been widely adopted, although it may be superseded later by its successor OWL.

From a knowledge-representational point of view ontologies can have the following components (e.g. [19]): concepts, relations and axioms. DAML+OIL supports the representation of all these kinds of components. Further, it has concept and relation constructors (e.g. Boolean operators and quantifiers) that allow for the definition of new concepts and relations based on already existing concepts and relations.

3. Related Work

Protégé-2000 is software for creating, editing and browsing ontologies. The design and development of Protégé-2000 has been driven by two goals: to be compatible with other systems for knowledge representation and to be an easy to use and configurable tool for knowledge extraction. Protégé-2000 is available as free software and should be installed locally. It also has a number of plug-ins, among others PROMPT, which is an algorithm for merging and aligning ontologies [18, 13]. When merging two ontologies, PROMPT creates a list of suggested operations. An operation can, for instance, be to merge two terms or to copy a term to the new ontology. The user can then perform an operation by choosing one of the suggestions or by specifying an operation directly. PROMPT performs the chosen operation and additional changes that follow from that operation. The list of suggestions is then updated and a list of conflicts and possible solutions to these conflicts is created. This is repeated until the new ontology is ready. PROMPT was previously called SMART and a high-level description of the algorithm is given in [12]. In [15] evaluation criteria for mapping or merging tools were proposed. First, an evaluation should be driven by pragmatic considerations: input requirements, level of user interaction, type of output and content of output. Tools that satisfy a user's pragmatic requirements can then be compared with respect to a performance criterion based on precision and recall. Protégé -2000 with PROMPT was evaluated according to these criteria.

The initial goal for developing Chimaera [3] was to provide a tool that could give substantial assistance for the task of merging knowledge bases produced by different users for different purposes with different assumptions and different vocabulary. Later the goals of supporting testing and diagnosing ontologies arose as well. The user interacts with Chimaera through a web browser [11]. The two main tasks when merging two ontologies in Chimaera are to merge two semantically identical terms from different ontologies so that they are referred to by the same name in the resulting ontology, and to identify terms that should be related via is-a, disjointness or instance relationships and provide support for introducing those relationships. Chimaera also supports the identification of the locations for editing and performing the edits. Today, Chimaera has support for merging taxonomies of concepts and for merging attributes. To assist the user Chimaera generates name resolution lists that suggest concepts that are candidates to be merged or to have taxonomic relationships not yet included in the merged ontology. Chimaera also generates a taxonomy resolution list where it suggests taxonomy areas that are candidates for reorganization. On the basis of these lists the user decides what should be done.

In [9] we evaluated how well PROMPT and Chimaera work for merging bio-ontologies. The ontologies we used for testing were Gene Ontology ontologies and Signal-Ontology. A larger survey on ontology tools and methodologies was performed by the OntoWeb Consortium [17].

4. SAMBO

The current implementation of SAMBO is a web-based system that helps a user merge two DAML+OIL ontologies into a new DAML+OIL ontology with unique names for terms (currently concepts or relations). The system separates the merging process into three steps: merging relations, merging concepts and introducing is-a relationships. Each step should be finished before the next step is started. In each step, the user can choose to manually merge terms or introduce is-a relationships in the ontologies or to have the system propose suggestions. The user can choose to accept or reject the suggestions. Upon acceptance of a suggestion, the system performs the actual merging or includes the is-a relationship, computes the consequences and makes the additional changes that follow from the operation. Upon rejection of a suggestion, it is checked that we do not have two different terms with the same name. If so, one of the terms needs to be renamed. In each case the suggestion list is updated. At each point in time the user receives information on which operations have been performed and how many suggestions there are left for the step. After all suggestions are processed or the user decides that no more merging should be performed or no more relationships between the ontologies should be added, the system copies the terms of the original ontologies that are not merged to the new ontology.

Regarding the generation of the suggestions we use the following strategies. For the first step (relations) the system generates a suggestion when the names of the terms in the different ontologies are the same or when one is an affix of the other. In steps two and three the SAMBO system gives the user the choice to use one of several underlying algorithms. This allows the user to experiment with different

strategies and obtain better results. Currently, we have implemented the following strategies.

1. N-gram string matching
2. Edit distance string matching
3. Linguistic matching + Porter stemming
4. Linguistic matching + Porter stemming + WordNet
5. Structure-based strategy

N-gram and edit distance matching are simple string matching algorithms. N-gram matching computes a similarity between strings based on n-grams. An n-gram is a set of n consecutive characters extracted from a string. Similar strings will have a high proportion of n-grams in common. In the second algorithm the similarity between strings is based on the edit distance. This is defined as the number of deletions, insertions, or substitutions required to transform one string into the other. The greater the edit distance, the more different the strings are. Algorithms three and four are linguistic matchers. We assume that the name of a term is represented as a string of words. The algorithm computes the similarity of the strings based on the similarity of the pairs of words. Within the algorithms we employ the Porter stemming algorithm. Algorithm four also uses WordNet to compute the similarity of the pairs of words. The matching thresholds, e.g. the similarity value in the linguistic matcher, can be modified by experts. In the structured-based strategy we use the structure of the ontologies and already merged concepts to propose new suggestions. In our implementation a path between two concepts is a composition of one or more is-a links in the is-a hierarchy. The user receives a list of already merged concept pairs and can ask the system to generate new suggestions based on the paths between two merged concepts. Thus, the strategy is based on the intuition that concepts between two given merged concepts in the is-a hierarchy have a good chance of being similar to each other.

In the second step (merging of concepts) the user can choose any of the strategies. A suggestion is generated based on the chosen strategy. We also make explicit use of the fact that several ontologies define a 'synonym' relation and thus check whether synonyms match the concept names.

In the third step (is-a relationships between concepts) the user can choose one of the linguistic matchers. The system generates a suggestion when a concept name in one ontology contains all the words of a concept name in the other ontology based on the linguistic matcher.

SAMBO uses further the FaCT [6] system to provide a number of reasoning services. (Our implementation is partly based on OilEd [2] ideas and implementation.) The user can ask the system to check whether the new ontology is consistent. She can receive information about unsatisfiable concepts and cycles in the ontology. The user can also receive an updated DAML+OIL ontology that is generated from the FaCT representation and therefore it contains less redundancy and it contains explicit statements for derived relationships.

5. Evaluation

We have performed two different evaluations. In the first evaluation we compare the quality of the suggestions that are generated by the different algorithms and strategies in our system. In the second evaluation we compare SAMBO with two well-known ontology merge tools regarding quality of the suggestions and the time it takes to merge two ontologies.

5.1. Comparison of algorithms

In this evaluation, we compare the quality of the suggestions generated by the different matching algorithms and strategies in our system.

Test Ontology

As test ontologies we have chosen two bio-ontologies that are available from Open Biological Ontologies (OBO) [16]. OBO is an umbrella address for structured shared controlled vocabularies and ontologies for use within the genomics and proteomics domains. The ontologies we chose are the Medical Subject Headings (MeSH) and the Anatomical Dictionary for the Adult Mouse (MA). MeSH is a controlled vocabulary produced by the American National Library of Medicine and used for indexing, cataloging, and searching for biomedical and health-related information and documents. It consists of sets of terms naming descriptors in a hierarchical structure. These descriptors are organized in 15 categories, such as the category for anatomic terms and the category for organisms. We used the MeSH category for anatomic terms, including approximately 1400 terms. MA is cooperating with the Anatomical Dictionary for Mouse Development (EMAP), to generate an anatomy ontology covering the entire lifespan of the laboratory mouse. The adult mouse anatomy ontology we used describes anatomical structures for the postnatal mouse (Theiler stage 28), including approximately 2350 terms. The ontology is represented as a directed acyclic graph. It organizes anatomical structures spatially and functionally, using is-a and part-of relationships. The two ontologies cover a similar subject domain, anatomy, and are developed independently. We translated the two ontologies from the GO flat file format to DAML+OIL retaining identifiers, names, synonyms, definitions, and is-a and part-of relationships.

Evaluation Result

Using the first linguistic matcher (using the Porter stemming algorithm, but not WordNet), our system found 377 suggestions. When also WordNet was used, 82 more suggestions are found. (WordNet has a good coverage of anatomy [1], so it was likely that new suggestions would be found.) These suggestions can be divided into four types. In the first type one term is the plural of the other (e.g. `ganglion` and `ganglia`). In the second type the terms are synonyms (e.g. `midbrain` and `mesencephalon`). These suggestions are useful. Further, there is the case where there is a semantic relationship between the terms. For instance, our system suggested `bile canaliculi` and `bile duct`. According to WordNet canaliculi is a kind of duct. This is a useful suggestion for is-a relationships. We

found that for every suggestion in this category, there always was an is-a or part-of relationship in the original ontologies. Finally, 12.5% of the suggestions from WordNet were due to different senses of words. For instance, `nerve` and `cheek` was suggested. These are synonyms in the sense of impudent aggressiveness, but not in the anatomical sense.

Using the N-gram matcher we obtained 661 suggestions (of which 370 were also found by the first linguistic matcher). As it is a flexible string matcher, it is able to find suggestions based on small differences in the spelling of words. For instance, it suggests `brain stem` and `brainstem`. It is able to find useful suggestions that are not found using WordNet, for instance, `striatum` and `neostriatum`. At the same time, 24.3% of the generated suggestions contain unrelated terms. For example, the suggestion `coeliac artery` and `iliac artery` refers to arteries but coeliac is of or relating to the abdominal cavity and iliac is of, relating to or located near the ilium. This matcher has also the characteristic that for a term in one ontology, often many similar terms in the other ontology are suggested for merging or is-a relationship. We found that for 53 concepts there were multiple suggestions. (For the 53 concepts together there were 165 suggestions.) For instance, for `hippocampus` the following were suggested: `hippocampus`, `hippocampus CA1` and `hippocampus CA2`. SAMBO displays these suggestions simultaneously.

Edit distance matching is a good approximate string matching algorithm that allows to eliminate spelling errors. In this test, however, it found the smallest number of correct suggestions.

When making use of the fact that these ontologies also contain information about synonyms, 9 more suggestions are found (e.g. `cerebellum lobule IX` and `uvula`).

We also experimented with the structure-based matcher with different path lengths. The maximum length of an is-a path in the MeSH anatomy ontology is 10, while for the MA ontology this is only 4. In this experiment we achieve the best result with maximum path length of 3. In this case we receive two new correct suggestions. A higher path length does not give more new results. One reason for the fact that only two correct new suggestions were found, is that, in our tests, most concepts in the paths were already merged. However, it may also be that the is-a relation alone does not define a good enough neighborhood of a concept to find useful suggestions. Further testing will make this clear.

5.2. Comparison of tools

In this evaluation we compare SAMBO with Protégé-2000 with PROMPT and Chimaera regarding the quality of suggestions and the time it takes to merge ontologies using these tools. We note that for Protégé-2000 with PROMPT and Chimaera we have used the systems as they are provided in the basic distribution. Extensions to the basic algorithms are being made. For instance, Anchor-PROMPT [14], an extension to PROMPT, uses paths between merged concepts in the two ontologies to find new suggestions.

Test Ontologies

We tested the tools using three 'cases' taken from two different domains: biological ontologies and ontologies about academia.

For the first two cases we used a part of an ontology from the Gene Ontology Consortium (GO) [5] together with a part from Signal-Ontology (SigO). The Gene Ontology Consortium is a joint project which goal is to produce a structured, precisely defined, common and dynamic controlled vocabulary that describes the roles of genes and proteins in all organisms. Currently, there are three independent ontologies publicly available over the Internet: biological process, molecular function and cellular component. The GO ontologies are becoming a de facto standard and many different bio-databases are today annotated with GO terms. The terms in GO are arranged as nodes in a directed acyclic graph, where multiple inheritance is allowed. The purpose of the SigO project is to extract common features of cell signaling in the model organisms, try to understand what cell signaling is and how cell signaling systems can be modeled. SigO is a publicly available controlled vocabulary of the cell signaling system. It is based on the knowledge of the Cell Signaling Networks databank [20] and treats complex knowledge of living cells such as pathways, networks and causal relationships among molecules. The ontology consists of a flow diagram of signal transduction and a conceptual hierarchy of biochemical attributes of signaling molecules. In our tests two cases were created. Each case consists of one part of SigO and one part of GO. Each case was chosen in such a way that there was an overlap between the GO part and the SigO part. The first case, *behavior* (B), contains approximately 60 terms from GO and approximately 10 terms from SigO. The second case, *immune defense* (ID), contains approximately 70 terms from GO and approximately 15 terms from SigO. We used more terms from GO than from SigO because the granularity of GO is higher than the granularity of SigO for these topics.

We also created a case, *academia* (AC), using two ontologies from the DAML ontology library. The first ontology describes an employment hierarchy in academic institutes. The ontology was developed at Carnegie Mellon. It contains 31 terms. The other ontology describes employees, publications and academic activities in a computer science department and was developed at the University of Maryland. It contains 83 terms. The overlap between these two ontologies is about employees in academic departments of a university.

Evaluation Results

In table 1 we show the results regarding the quality of the generated suggestions in terms of precision and recall for the cases *behavior* (B), *immune defense* (ID) and *academia* (AC), respectively. (The numbers for PROMPT and Chimaera for the B and ID cases are taken from [9].) Precision measures how many of the suggestions are relevant while recall measures how many of the relevant suggestions the system actually proposed. SAMBO performs perfect with respect to quality of suggestions on the *behavior* case and good on the *immune defense* case. However, it performs worse than PROMPT in the *academia* case.

We can have a look at the experiments for SAMBO in more detail. Tables 2 and 3 show the suggestions created by SAMBO (N-gram) for the *immune defens*e case and the *academia* case, respectively. The first column describes whether the suggestion was for a merge (M) or is-a relationship (R). The last column describes whether the

suggestion was correct (C), wrong (W) or not found by the system (NF). In the *immune defense* case, the two missing suggestions are for is-a relationships between concepts. They might have been found by using more flexible string matching. However, more flexible string matching may also lead to more wrong suggestions. The wrong suggestions in *immune defense* have to do with similar concepts where the matching was too flexible. As we mentioned before, if a concept from the first ontology is involved in multiple suggestions, the user receives these multiple suggestions simultaneously. Further, when the user has decided to merge, the suggestion list is updated and therefore not all the wrong suggestions will actually be shown to the user. In the *academia* case there are two missing suggestions for merging and the system did not find any correct suggestions for the is-a relationships. The names of the terms in the two ontologies differed too much for the algorithm to find them.

Based on these tests the quality of SAMBO's merging suggestions is good. However, the system behaves from good to poor when suggesting is-a relationships. This can be explained by the fact that we use linguistic and string matching. Linguistic matching and string matching find concepts that are similar in name. This works for some examples (e.g. `Antigen Processing and Presentation` is-a `antigen presentation`), but there are also many examples for which this strategy does not work (e.g. `fever` is-a `inflammation`).

Table 4 illustrates the results regarding the time it took to merge the test ontologies. The total work time is computed as the work time based on the suggestions plus the time for merging and adding the relationships that the systems missed to propose. Due to the good quality of suggestions compared to the other systems as well as the easy to use user interface and the automatic copy operations, working with SAMBO was much faster than working with the other two systems.

Tool	Case	Sug.	Correct	Missing	Recall	Precision
PROMPT	B	3	3	2	0.6	1
Chimaera	B	16	4	1	0.8	0.25
SAMBO	B	5	5	0	1	1
PROMPT	ID	4	4	5	0.444	1
Chimaera	ID	10	4	5	0.444	0.4
SAMBO (N-gram)	ID	9	7	2	0.777	0.777
SAMBO (edit distance)	ID	9	7	2	0.777	0.777
SAMBO (ling. matcher)	ID	8	7	2	0.777	0.875
SAMBO (ling. matcher, WordNet)	ID	9	7	2	0.777	0.777
PROMPT	AC	7	5	6	0.454	0.714
Chimaera	AC	1008	8	3	0.727	0.008
SAMBO (N-gram)	AC	10	5	6	0.454	0.5
SAMBO (edit distance)	AC	10	5	6	0.454	0.5
SAMBO (ling. matcher)	AC	9	4	7	0.363	0.444
SAMBO (ling. matcher, WordNet)	AC	14	4	7	0.363	0.285

Table 1. Quality of suggestions

	Gene Ontology	Signal Ontology	
M	synonym	synonym	C
M	immune response	Immune Response	C
M	B-cell activation (synonym: B-cell proliferation)	B Cell Activation	C
M	B-cell activation (synonym: T-cell proliferation)	T Cell Activation	W
M	T-cell activation (synonym: T-cell proliferation)	T Cell Activation	C
M	T-cell activation (synonym: T-cell proliferation)	B Cell Activation	W
M	complement activation	Complement Signaling (synonym: complement activation)	C
R	antigen processing	Antigen Processing and Presentation	C
R	antigen presentation	Antigen Processing and Presentation	C
R	*inflammatory response*	*Inflammation*	NF
R	*activation of natural killer cell activity*	*Natural Killer Cell Response*	NF

Table 2. Suggestions for *immune defense* with N-gram

	Academic Ontology of UM	Academic Ontology of CMU	
M	Student	Student	C
M	Administrative Staff	AdministrativeStaff	C
M	Director	Director	C
M	Faculty	Faculty	C
M	Professor	Professor	C
M	Assistant (includes research and teaching assistants)	Assistant (is a kind of administrative staff)	W
M	*Postdoctoral Fellow*	*PostDoc*	NF
M	*Technical Staff*	*Systemstaff*	NF
R	Research Programmer	Research	W
R	Research Staff	Research	W
R	Research Scientist	Research	W
R	Research Engineer	Research	W
R	*Masters Student*	*GraduateStudent*	NF
R	*PhD Student*	*GraduateStudent*	NF
R	*Research Staff*	*ResearchAssistant*	NF
R	*Visiting Staff*	*VisitingProfessor*	NF

Table 3. Suggestions for *academia* with N-gram

Tool	Case	Work based on suggestions	Total work time
PROMPT	B		15
Chimaera	B	8	8
SAMBO	B	3	3
PROMPT	ID		28
Chimaera	ID	10	15
SAMBO	ID	4	6
PROMPT	AC		20
Chimaera	AC	18	22
SAMBO	AC	3	7

Table 4. Time of Merging Process

6. Conclusion

We have described SAMBO, a merging tool for DAML+OIL ontologies, evaluated the suggestions generated by its different algorithms and strategies and compared the system in terms of quality of suggestions and work time with PROMPT and Chimaera. Regarding the quality of suggestions SAMBO did well compared to the others and outperformed the other tools regarding time. We note that our test ontologies did not require the full expressivity of DAML+OIL. Therefore, the full description logic reasoning capabilities of the tool have not been exploited yet in this test. We will also investigate further on more advanced strategies for the generation of suggestions.

The system still needs to be evaluated with respect to other factors as described in e.g. [9] and [10]. For instance, we already know that the user interfaces of Protégé-2000 and Chimaera have a more elaborate visualization of the ontologies than our current implementation. Also, these systems allow for many different input and output formats and are more full-fledged ontology engineering tools than our system.

Acknowledgements

We thank Vaida Jakoniene and Bo Servenius for comments on the system. We also acknowledge the financial support of the Center for Industrial Information Technology and of the EU Network of Excellence REWERSE (Sixth Framework Programme project 506779).

References

[1] Bodenreider, O., Burgun, A. Characterizing the Definitions of Anatomical Concepts in WordNet and Specialized Sources. Proceedings of the First Global WordNet Conference, pp 223-230, 2002.

[2] Bechhofer, S., Horrocks, I., Goble, C., Stevens, R. OilEd: a Reason-able Ontology Editor for the Semantic Web. Proceedings of the Joint German Austrian Conference on Artificial Intelligence, LNAI 2174, pp 396-408, Vienna, Austria, 2001.

[3] Chimaera. http://www.ksl.stanford.edu/software/chimaera/

[4] DAML+OIL. http://www.w3.org/TR/daml+oil-reference

[5] The Gene Ontology Consortium. Gene Ontology: tool for the unification of biology. Nature Genetics, 25(1):25-29, 2000. http://www.geneontology.org/.

[6] Horrocks, I. FaCT and iFaCT. Proceedings of the International Workshop on Description Logics, pp 133-135, Linköping, Sweden, 1999.

[7] Horrocks, I. DAML+OIL: a Reason-able Web Ontology Language. Proceedings of the International Conference on Extending Database Technology - EDBT02, LNCS 2287, pp 2-13, Prague, Czech Republic, 2002.

[8] Lambrix, P. Ontologies in Bioinformatics and Systems Biology. Chapter in Dubitzky, W. and Azuaje, F. (eds.) Artificial Intelligence Methods and Tools for Systems Biology, Kluwer, to appear, 2004.

[9] Lambrix, P., Edberg, A. Evaluation of ontology merging tools in bioinformatics. Proceedings of the Pacific Symposium on Biocomputing - PSB03, pp 589-600, Kauai, HI, USA, 2003.

[10] Lambrix, P., Habbouche, M., Pérez, M. Evaluation of ontology engineering tools for bioinformatics. Bioinformatics, 19(12):1564-1571, 2003.
[11] McGuinness, D., Fikes, R., Rice, J., Wilder, S. An Environment for Merging and Testing Large Ontologies. Proceedings of the Seventh International Conference on Principles of Knowledge Representation and Reasoning - KR2000, pp 483-493, Breckenridge, Colorado, USA, 2000.
[12] Noy, N.F., Musen, M. SMART: Automated Support for Ontology Merging and Alignment. Proceedings of the 12th Workshop on Knowledge Acquisition, Modeling and Management, Banff, Canada, 1999.
[13] Noy, N.F., Musen, M. PROMPT: Algorithm and Tool for Automated Ontology Merging and Alignment. Proceedings of Seventeenth National Conference on Artificial Intelligence, pp 450-455, Austin, TX, USA, 2000.
[14] Noy, N.F., Musen, M. Anchor-PROMPT: Using Non-Local Context for Semantic Matching. Proceedings of the IJCAI01 Workshop on Ontologies and Information Sharing, Seattle, WA, USA, 2001.
[15] Noy, N.F., Musen, M. Evaluating Ontology-Mapping Tools: Requirements and Experience. Proceedings of the EKAW Workshop on Evaluation of Ontology Tools, Siguenza, Spain, 2002.
[16] Open Biological Ontologies. http://obo.sourceforge.net
[17] OntoWeb Consortium. Deliverable 1.3: A survey on ontology tools. 2002. http://www.ontoweb.org
[18] Protégé with PROMPT. http://protege.stanford.edu/index.html
[19] Stevens, R., Goble, C., Bechhofer, S. Ontology-based knowledge representation for bioinformatics. Briefings in Bioinformatics, 1(4):398-414, 2000.
[20] Takai-Igarashi, T., Nadaoka, Y., Kaminuma, T. A Database for Cell Signaling Networks. Journal of Computational Biology, 5(4):747-754, 1998.

SCIENTIFIC PAPERS UNIVERSITY OF LATVIA. 2004. VOLUME 672, Datbases and Information Systems
436–450 P.

Combining FCA and a Logic Language for Ontology Representation

Hele-Mai Haav

Institute of Cybernetics at Tallinn University of Technology,
Akadeemia tee 21, 12618 Tallinn, Estonia
helemai@cs.ioc.ee

Abstract. The most important problem of ontology design is to guarantee accuracy, transparence and consistency of ontology representation. This paper provides automatic way of transforming lattice based ontology representation to logical expression by defining first order logic model of concept lattice based ontology expression. As the latter represents only taxonomic relationships between concepts, then ontology designer is given possibility to add additional concepts, properties, and ontological relationships using a rule language based on first order logic. Validation of ontology, reasoning about ontology, and search is done using logical inference.

Keywords: Ontology representation, Formal Concept Analysis, logic languages

1. Introduction

The subject of ontology is the study of the categories of things that exist or may exist in some domain of interest [Sowa 2000].

Formal ontologies provide machine-processable semantics of information that is shared between different agents (humans or software). Ontologies play important role in many application domains including semantic web services, business systems, industrial systems, intelligent information retrieval, etc.

As analysis of existing ontologies and ontology design methodologies shows, it is very difficult for a designer to develop accurate, transparent and consistent ontology [Tempich and Volz 2003, Fernandez-Lopez and Gomez-Perez 2002, Maedche 2002]. Domain ontology design usually involves work of domain experts, software engineers and ontology designers. All parties need to reach to design agreement.

In order to help experts in ontology design process, an initial domain ontology can be automatically or semi-automatically built from domain-specific knowledge captured in domain-specific texts, documents, or data [Gangemi et al 1999, Maedche and Staab 2000, Haav 2003]. Ontology designer can do further development of the initial ontology.

Next step after extraction of the initial ontology is to formally express ontology in some logic language for sharing knowledge by software agents; ontology based

reasoning purposes and search. This can also be done automatically or semi-automatically.

The main goal of this paper is to present a model of concept lattice based ontology expression in the form of first order language in order to make it possible to represent in addition to taxonomic relationships also non-taxonomic relationships between the concepts. This work is continuation and extension of our previous work on semi-automatic extraction and expression of domain-specific ontologies using Formal Concept Analysis (FCA) [Haav 2003]. In our previous approach, an initial representation of domain ontology is extracted from a set of domain-specific texts as a concept lattice using FCA and NLP [see Haav 2003]. As an extension to this work, our current approach presented in this paper, takes concept lattice based ontology expression and maps it automatically to a set of rules (and facts) in first order language. This is done according to the first order language model of concept lattice based ontology description. As concept lattice based ontology description represents only taxonomic relationships between concepts, then ontology designer is given possibility to add additional concepts and relationships (part-of, related to, etc) using a rule language based on first order logic. Validation of ontology, reasoning about ontology and search can be done using logical inference. As our approach uses first order language, then it is possible to attach different ontology inference engines for practical applications by translating ontology expression to any inference engine rule language.

Our approach is applicable in many application domains, where domain specific ontologies can be extracted form web catalogs, product catalogs, domain specific dictionaries and texts etc.

The rest of the paper is structured as follows. Section 2 gives a motivation for the approach and refers to the related works. General framework of proposed approach is presented in section 3. First order language model of concept lattice based ontology expression is given in section 4. Section 5 shows how a rule language can be used to present non-taxonomic relationships. Section 6 concludes the work.

2. Motivation and related work

Recent study of DAML ontology library by Tempich and Volz [Tempich and Volz 2003] shows that semantic web ontologies are designed in rather heterogeneous way and many semantic web ontologies fail in being usable for inference. This indicates that quality of ontology representation is not high enough.

Also analysis and evaluation of ontology development methodologies [Fernandez-Lopez and Gomez-Perez 2002] result in conclusion that high level design methods do not support well nontrivial ontology-based reasoning. The latter is an important feature of any ontology development methodology, because an ontology expression created by human experts can easily be inconsistent.

Our motivation in this paper is to provide assistance to ontology designer in ontology design process in order to guarantee accuracy, transparence and consistency of ontology representation by automatic or semiautomatic methods of ontology extraction and expression.

Related work on this topic can be grouped as follows:

- Work on extraction of taxonomic relationships and concept hierarchies from given text or data
- Work on formally representing intended semantics of ontology description in some logic language

We have been inspired by Formal Concept Analysis (FCA) [Ganter and Wille 1999] as one of the methods of learning concept taxonomies. FCA algorithmically constructs concept lattice from binary relationship between objects and their attributes. FCA is used in ontology engineering for merging ontologies in FCA-MERGE method [Stumme and Maedche 2001]. There are other methods available for extraction of taxonomic relationships as hierarchical clustering techniques [Manning and Schuetze 1999] or pattern-based approaches [Hobbs 1993, Maedche 2002]. Also works on extraction of concept hierarchies are of interest. Some related works can be found in [Assadi 1999, Hofmann 1999].

There are many approaches developed within Semantic Web community in order to define mapping of RDF schemas to some of logical languages for providing formal semantics [Maedche 2002]. Most widely used are description logic [Baader et al. 2002], first-order logic and F-logic [Kifer et al 1995]. Grüninger and Fox [Grüninger and Fox 1995] provide ontology development methodology (TOVE project), which allows to manually transform informal natural language specifications into computable model expressed in first order logic. $(KA)^2$ ontology uses F-logic, also Ontobroker [Decker at al 1999] and Text-To-Onto [Maedche 2002] systems use F-logic for inference.

To the best of our knowledge, we do not know works, which report about direct automatic or semi-automatic extraction of logical descriptions of a domain ontology.

In the approach described in this paper, we also do not extract logical expressions of domain ontology but instead we rely on initial ontological structure in the form of a lattice learned from domain-specific texts. We provide automatic way of transforming lattice based ontology expression to logical expression by defining first order logic model of concept lattice based ontology expression.

3. Concept lattice based ontology expression

3.1. Formal Concept Analysis

In this section, Formal Concept Analysis (FCA) developed by Ganter and Wille [Ganter and Wille 1999] is very briefly introduced in order to give basis for understanding our approach.

FCA is a result of an attempt to restructure mathematical order theory and lattice theory and as such gives a new interpretation of complete lattices as concept lattices. Currently, FCA can be seen as a field of applied mathematics used in a number of applications of conceptual analysis [Godin 1991], knowledge representation [Stumme and Maedche 2001], data analysis, etc.

FCA allows extraction of similar groups of objects from a set O of objects described by a set of attributes C using binary relationship R on OΔC. For o$\subset$O and c$\subset$C we note R(o,c)=1 if oRc (the object o has the attribute c, the relationship R holds) and R(o,c)=0 if o$\not$ Rc respectively.

The table K(O,C,R) is called formal context that can be used to extract groupings and relationships between objects and attributes. In the framework of FCA the discovered groups represent the closed set of the Galois connection induced by R on the pair O and C. Two mappings are defined as follows.

If X is an arbitrary part of O, then we may define mapping f that maps X onto the set of all elements of C that are related to all elements of X as follows:

$$f(X) = \{y \subset C \surd \&x \subset X: xRy\}$$

Similarly, if Y is an arbitrary part of C, then we may define the following mapping:

$$g(Y) = \{x \subset O \surd \&y \subset Y: xRy\}$$

These mappings mean that f(X) is the set of all attributes shared by all objects in X and g(Y) is the set of all objects that have all attributes in Y.

The mappings f and g are monotonously decreasing in the following sense:

$$X \geq X' \blacktriangledown\ f(X') \geq f(X)$$
$$Y \geq Y' \blacktriangledown\ g(Y') \geq g(Y)$$

The pair of mappings (f, g) meets the criteria of Galois connection between the powerset P(O) and the powerset P(C). The closed subsets of both O and C form two lattices wrt the set inclusion. Hence, we have two lattices L_O and L_C that are isomorphic. Consider the set L of all pairs of corresponding parts of lattices L_O and L_C so that each element of this set is the Cartesian product of closed parts of O and C (i.e., $X \Delta f(X)$ or $g(Y) \Delta Y$ wrt R).

These pairs are called *formal concepts* by Wille [Ganter and Wille 1999]. In FCA, X is referred to as the concept extent and Y as the concept intent. Thus, a formal concept is a pair of a set of objects that have common attributes (the extent), and the defining set of attributes that they have in common (the intent). Each concept is uniquely determined by either its extent or intent. The partial order relation on the set of all formal concepts of the given context K(O,C,R) is determined by the set inclusion between the extensions or equivalently by the reverse inclusion between intensions of concepts.

In this paper, we define the partial order relation on L based on the set inclusion between the extensions of concepts as follows,

$$X \Delta Y\ \Omega X' \Delta Y' \text{ iff } X \geq X' \text{ and } Y' \geq Y.$$

This lattice L is called the Galois lattice or formal concept lattice of the relationship R on $O \Delta C$. This partial order defines subconcept and superconcept relationships. Concept lattice is a complete lattice [Ganter and Wille 1999] meaning that for each set of formal concepts, there is always a greatest lower bound (glb or greatest common subconcept) and a least upper bound (lub or least common superconcept). Given an object o, there is always a most specific concept, whose extent contains o, this concept is called *object concept*. Dually, for each attribute c, there is a most general concept whose intent includes c, this is called *attribute concept*.

FCA is an algorithmic method for construction of concept lattices for a given context. There are several algorithms for generating the set of all formal concepts and construction of line diagrams of concept lattices. Excellent comparison of performance of those algorithms can be found in [Kuznetsov and Objedkov 2001].

3.2. Extraction of a concept lattice

In this section we briefly overview our method [Haav 2003] used to extract a concept lattice from domain-specific texts. This is a basis of our new approach presented in this paper.

According to the method, first task is to produce a formal context K(O,C,R) for extractable domain ontology. As a set of objects O, domain-specific text sources are considered. A set of noun-phrases from the texts is taken as a set of attributes C. It is assumed that noun-phrases denote/indicate concepts used in application domain, as text sources are domain specific and use specific vocabulary. Binary relationship R between descriptions (texts) of domain entities and noun phrases is discovered during the NLP process of text sources.

Next step is to perform FCA on the context. As a result, formal concept lattice that corresponds to the domain ontology is constructed.

For our current approach presented in this paper, the resulting lattice is reduced and certain naming procedure is performed in order to use it as concept lattice based ontology expression (see section 3.4 below). In the following sections, small examples from real estate domain are provided in order to illustrate FCA and its usage in our approach.

3.3. A real estate domain example

As an example, consider real estate catalogs. For example, Table 1 represents formal context for real estate domain. For each description of a real estate item, there is a corresponding text source (e.g. A1 corresponds to text about real-estate item one, etc.). Noun-phrases are extracted from each catalog entry by using NLP tools. In the table below, existence of a relationship between a catalogue entry (text) and a noun-phrase is denoted by 1. We feel free to display only some noun-phrases chunked from real estate catalog entries. This is in order to obtain a small sample lattice in the example. In this paper, we are not interested in how noun-phrases are extracted from the texts but we concentrate to the formal context obtained from the texts and presented as in the Table 1 below.

Table 1. Real estate domain context

Objects	**Attributes (Noun-phrases)**					
	Real estate	Family house	Country house	Summer house	Blockhouse	Apartment
A1	1	1				
A2	1		1			
A3	1		1	1		
A4	1				1	1

After applying FCA to this context we get a set of formal concepts of this given context as follows.

1. {{A1}, {real estate, family house}}
2. {{A2, A3}, {real estate, country house}}
3. {{A3}, {real estate, country house, summerhouse}}
4. {{A4}, {real estate, blockhouse, apartment}}
5. {{A1, A2, A3, A4}, {real estate}}
6. {{Ø}, {real estate, family house, country house, summerhouse, blockhouse, apartment}}

The graphical representation of the concept lattice is depicted in the form of Hasse diagram as following Fig. 1.

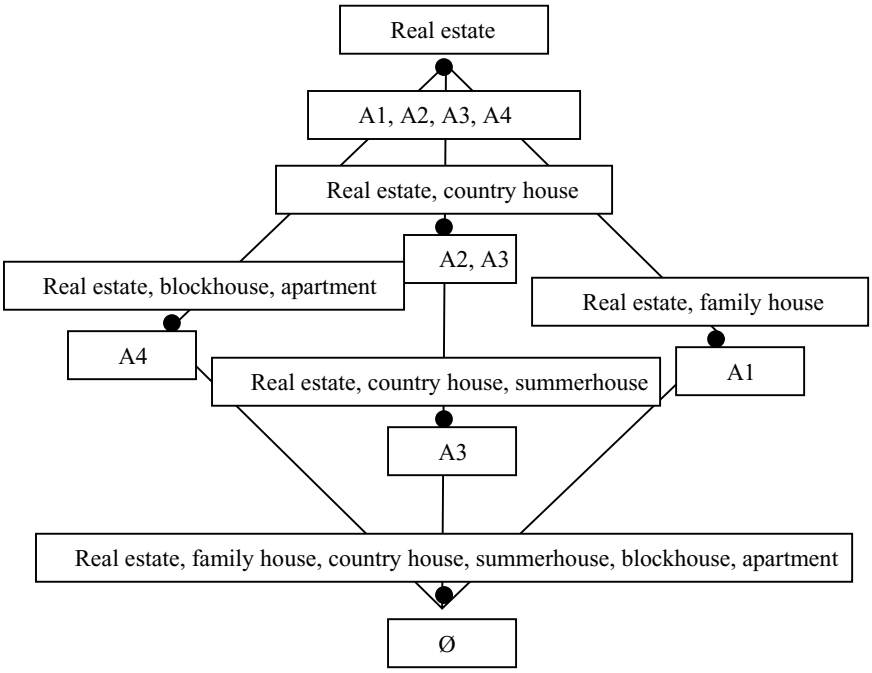


Figure 1. Concept lattice of real estate domain

Each node in this lattice (denoted by black circle) is a formal concept. For example, one of the formal concepts of the context described in Table 1 is as follows:

{A1, A2, A3, A4,} Δ *{Real estate},* where the set *{A1, A2, A3, A4,}* is the extent of the concept and the set *{Real estate}* is its intent. This is top element of the lattice. This is most general concept; it has one attribute that is shared by all real estate catalog entries. The bottom element of the lattice (lattice bottom) is the least general concept that is defined by all the attributes at the same time. In our example, there

are no objects that are defined by all the attributes, so the extent of the bottom concept is empty.

Sub and super-concept relationships between the formal concepts are represented by edges in the Hasse diagram in Fig. 1. For example, the formal concept *{A3}* Δ *{Real estate, Country house, Summerhouse}* is a sub-concept of the concept *{A2, A3}* Δ *{Real estate, Country house}*.

Inheritance of attributes is also present in the concept lattice according to sub and super-concept relationships.

3.4. Concept lattice based ontology expression

In this section, we define concept lattice based ontology expression. There might arise confusion in using of the word concept in ontology research comparing to FCA. There is no direct notion in ontology field to denote formal concepts. Ontology concepts can be compared to FCA attributes, as both can be considered as unary predicates on the set of objects.

Our idea in creating concept lattice based ontology expression is to use duality feature of concept lattice, i.e. lattice of intensions and lattice of extensions of concepts are connected via Galois connection. In principle, in our case we consider only lattice of intensions of concepts as a useful structure for ontology expression learned during the concept lattice construction process.

There is redundant information in concept lattice. For a formal concept C= (X, Y), X will be present in every ancestor of C and symmetrically, Y will appear in every descendant. The two kinds of redundancy can be eliminated from concept lattice without losing any information as shown in [Godin 1991]: redundant attributes in formal concepts intents and redundant objects in formal concepts extents.

Our reduction procedure has 2 steps: elimination of redundant attributes from full concept lattice and elimination of lattice of extents.

Elimination of redundant attributes. Let Y′ be the set of elements of Y (intent) that do not appear in any descendant of Y. To eliminate redundant attributes we define a pair (X, Y) as a pair (X, Y'), where $Y' = \{y \subset C \surd g(y) = X\}$. Taking union of the Y' sets for the ancestors of a pair (X, Y'), including this pair itself recovers the initial pair (X, Y).

Elimination of lattice of extents. From the reduced lattice above, we eliminate lattice of extents L_O and get reduced lattice of intents L_{CR} of formal concepts.

We call the resulting lattice L_{CR} of reduction procedure as concept lattice based ontology expression. Fig. 2 shows the lattice L_{CR} of our example.

There are 3 aspects that need to be clarified as follows:

1. The resulting lattice is dependent on the context. Whenever we change formal context, we get a new lattice structure. For learning initial domain lattice we need to go through concept lattice construction procedure several times using different formal contexts (e.g. Real Estate Catalog entries in our example).

2. After reducing concept lattice to its intentional part, we need to give formal concepts the names. Naming in our case can be done as follows:
   a. A concept gets a unique name that is the name of the attribute(s) of formal concepts, which are left after reduction procedure. For each attribute c, there is a most general concept whose intent includes c, this is called *attribute concept* and the name of this concept is the name of corresponding attribute(s). Let us recall that attributes of formal concepts indicate domain specific concepts in our approach.
   b. After the previous naming procedure, there might be nodes that do not get names. In principle, the names for these nodes need to be provided by domain expert or ontology designer. It is possible automatically generate formal names (e.g. c1, c2…) for those nodes and then ask advice from human expert.
3. Human expert should manually add concept descriptions in natural language to each concept to express meaning of a concept in application domain.

In order to illustrate what was said above the Fig. 2 depicts real estate domain ontology produced from concept lattice shown in Fig.1 using reduction and naming procedures.

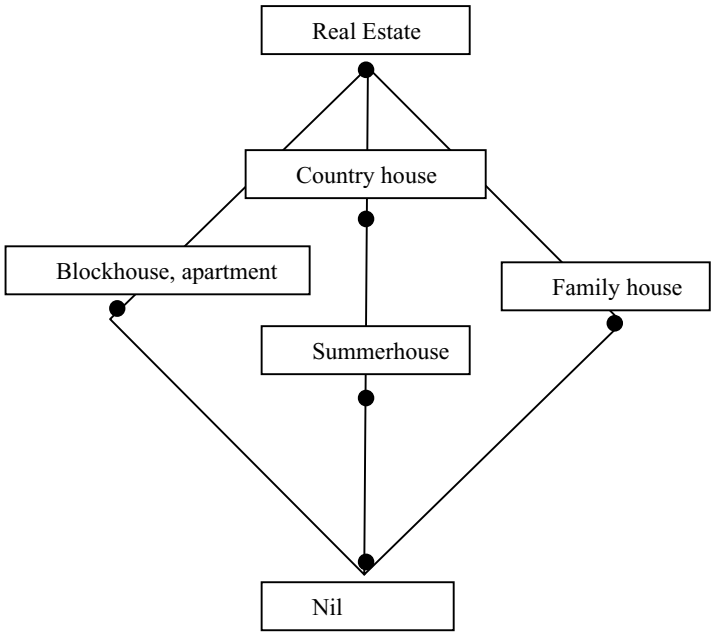


Figure 2. Reduced concept lattice and naming of concepts

This lattice displays names of concepts according to reduction and naming procedure. One concept has combined name: blockhouse-apartment. This indicates that there might be non-taxonomic relationship between the concepts or these could be separate concepts but a given formal context was not complete enough to enable

to extract them. The bottom element of the lattice is empty, which is denoted by generated name Nil.

The lattice does not refer to objects forming an extent of a certain concept but these can be recovered if necessary.

The following is concept lattice based ontology expression (see Fig. 3 below) of the same domain but another formal context is used. It illustrates the naming of concepts that do not get name by the naming procedure.

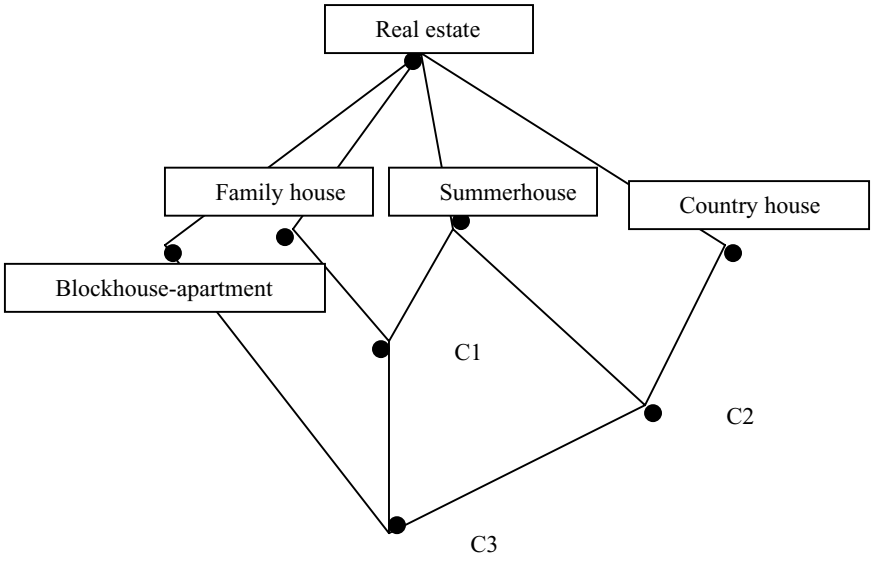


Figure 3. Naming of concepts

The nodes denoted by generated concept names C1, C2, C3 do not have labels in the lattice. Ontology designer may analyse the lattice above and find out that concept C1 is sub-concept of both concepts: Family house and Summerhouse. C1 then denotes the concept that is a family summerhouse and C2 denotes a country summerhouse respectively. The naming is up to ontology designer in this case.

In conclusion, our approach is about learning domain specific ontology from extensionally defined collections of instances contained in data (e.g. real estate catalog). It is assumed that the instances represent relevant knowledge for inductive extraction of intensional descriptions for a given domain.

Our next goal is to define first-order logic model for concept lattice based ontology expression in order to provide richer language for representation of ontological relationships and specify intended meaning of the descriptions.

4. First order logic model of concept lattice based ontology expression

In the previous section, we have defined domain ontology as a concept lattice reduced to its intensional part L_{CR} (see for example Fig. 3). In this section, we provide first order logic model for L_{CR}. At the moment we are not interested in extensions, which gave birth to full concept lattice for a formal context of a domain. In principle, it might be interesting to take extensions also into account when building a logic model, but it is out of scope of this work as we are concentrated to reasoning about concepts rather than their extensions. Nevertheless, existence of full concept lattice gives always an opportunity to find extent of a given concept.

In order to build first order logic model for concept lattice based ontology expression we need to define mappings from lattice structure to a first order language.

4.1. Language constructs

We use standard syntax for first order logic and define a simple rule language based on Horn clauses as follows.

An alphabet of the rule-language is defined as follows:

1. Set of constants **N** that consists of the set of concept names **C**, names of properties A, and special names **any** (lattice top, empty top is always True) and **nil** (lattice bottom, empty bottom is always False).
2. Set of variable names **V**. Uppercase letters denote the variables in **V**.
3. Set of predicate symbols **P**

Terms are either constants or variables. An atom (atomic formula) is a formula of the form $p(t_1,...,t_n)$, where p is a predicate symbol and $t_1,...,t_n$ are terms. A formula is called ground if it contains no variables.

Horn rules (clauses) have at most one atom on its head and they are formulas of the following form:

$$A \Uparrow B_1, B_2,...,B_n ,$$

where $B_1, B_2,...,B_n$ is conjunction of atoms B_i, i=0,...,n and universal quantification of variables is assumed.

A definite clause has exactly one atom in the head. A ground rule contains no variables. A rule with an empty head is considered to be a query or constraint. Ground rule with an empty body is called a fact.

Rules are used to define conceptual relationships and operations on lattice. Inference rules based on concept lattice are predefined.

In addition, users are also provided to specify constraints or any other ontological relationships in the form of the rules.

An interpretation for the rule-language is defined as a set of ground atoms constructed from predicate names in **P** and constants in **N.** As the language is general, then different inference engines can be used. For example, Java Expert Systems Shell (Jess) [Jess] or Gandalf [Tammet 1997] can be used as inference engines.

4.2. Mappings of concept lattice to rule language

The mappings from lattice to rule language are defined as follows.

Mapping concepts
Concepts are represented using their names in ontology as constants in rule language. For example, house, summerhouse etc. If we like to refer to extents, then concepts can also be represented by predicates like house(X).

Mapping taxonomic relationships
Predicate isa is used to represent partial order relationship between concepts. For example, the predicate isa(summerhouse, real-estate) defines partial order relationship between the concepts summerhouse and real-estate stating that *summerhouse* isa *real-estate* (i.e. summerhouse is subconcept of realestate). The predicate subconcept(Concept1,Concept2) is used to denote that *Concept1* is an immediate subconcept of *Concept2*. Subconcept predicates are automatically generated according to the given lattice L_{CR}.

Rules for lattice axioms
As L_{CR} is complete lattice, then the rules for lattice axioms are as follows:

Reflexivity:
isa(Concept, Concept)

Transitivity:
isa(Concept1, Concept2)⇑ subconcept(Concept1, Concept2)
isa(Concept1, Concept2) ⇑ isa(Concept1, Concept3), isa(Concept3, Concept2)

Predicate subconcept denotes that Concept1 is an immediate subconcept of Concept2.

Antisymmetry:
equal(Concept1,Concept2) ⇑ isa(Concept1,Concept2), isa(Concept2,Concept1)

Rules for lattice operations
As L_{CR} is a complete lattice, then for each set of concepts, there exists always a greatest lower bound (glb or greatest common subconcept) and a least upper bound (lub or a least common superconcept). Lattice meet is used to calculate glb and join is operation to calculate lub. We define these operations using the following set of rules.

Meet operation

common_subconcept(C, C_1,…,C_k)⇑ isa(C, C_1),…, isa(C, C_k)

greatest_common_subconcept(C, C_1,…,C_k) ⇑ common_subconcept(C, C_1,…,C_k),
common_subconcept(T, C_1,…,C_k),
isa(T, C)

The predicate common_subconcept(C, C_1,…,C_k) means that the concept C is a common subconcept of the set of concepts {C_1,…,C_k}

The predicate greatest_common_subconcept(C, C_1,…,C_k) means that the concept C is the greatest common subconcept of the set of concepts {C_1,…,C_k}.

Symmetrically, we define predicates and rules for join operation as follows:

Join operation

common_superconcept(C, C_1,…,C_k)⇑ isa(C_1, C),…, isa(C_k, C)

least_common_superconcept(C, C_1,…,C_k) ⇑
common_superconcept(C, C_1,…,C_k),
common_superconcept(T, C_1,…,C_k)
isa(C, T)

Logical model of a given lattice L_{CR} can automatically be generated on the basis of mappings presented above. This process is demonstrated in the following example.

4.3. Examples about mappings and inference

On the basis of concept lattice based ontology expression shown in Fig. 3 the following set of ground subconcept atoms is generated.

subconcept(familyhouse, real-estate)
subconcept(summerhouse, real-estate)
subconcept(countryhouse, real-estate)
subconcept(blockhouse-apartment, real-estate)
subconcept(c1,familyhouse)
subconcept(c1,summerhouse)
subconcept(c2,summerhouse)
subconcept(c2,countryhouse)
subconcept(c3,blockhouse-apartment)
subconcept(c3,c1)
subconcept(c3,c2)

Inference rules for lattice axioms and operations can be used to decide taxonomic relationships between concepts as well as perform lattice operations.

For example, to find the least common superconcept of the set of concepts *{summerhouse; countryhouse},* we define the following query:

least_common_superconcept(X, summerhouse, countryhouse).

The answer is the concept *real-estate*. If we are interested in finding the greatest common subconcept of the concepts *familyhouse* and *summerhouse*, then the

corresponding query is as follows: greatest_common_subconcept(X, familyhouse, summerhouse). The answer is the concept *c1*.

We may be interested in all the superconcepts of the concept *familyhouse*, for example. The query isa(familyhouse, X) gives the list of ground atoms as an answer. In our example, this is one atom isa(familyhouse, real-estate).

5. Representing non-taxonomic relationships

As we have seen, taxonomic relationships between concepts can be automatically generated from a given lattice based ontology expression L_{CR}. It is easy to add more rules and facts to the ontology expression above. In order to define non-taxonomic relationships the corresponding groups of predicates and rules should be defined.

Properties of concepts

For defining properties of concepts, the following predicate can be used:
hasproperty(Conceptname, Propertyname).

For example, the following ground atom can be added to the sample ontology description above: hasproperty(real-estate, location).

Inheritance of properties

Inheritance of properties can be represented by the following rule:

hasproperty(C1, X)⇑ isa(C1,C2), hasproperty(C2, X)

We may ask the query: hasproperty(countryhouse, X) and receive the answer that it has also property *location* as the concept *real-estate.*

Ontological relationships

Ontological relationships like part-of, related-to etc can be easily represented via predicates. The following predicates demonstrate opportunities adding other ontological relationships:

partof(C1,C2)
related(C1,C2)
synonyms(C1,C2), etc.

For example, according to predicates above, ontology designer can add the following ground atoms to the ontology model:

partof(apartment, blockhouse)
relatedto(blockhouse, city)
synonyms(familyhouse, familyhome)

Additional specific inference rules can easily be added to the set of predefined lattice based inference rules.

In the end, non-taxonomic relationships give additional possibilities for ontology representation and reasoning about ontology.

6. Conclusion

We have shown how concept lattices can be used for ontology representation and we have stated notion of concept lattice based ontology expression.

We defined a Horn clause model of concept lattice based ontology expression in order to enhance ontology expression with descriptions of non-taxonomic relationships between concepts. According to this model, a given lattice based ontology expression can automatically be transformed to logical expression in first-order language.

Validation of ontology, reasoning about ontology and search can be done using logical inference. As our approach uses first-order language, then it is possible to attach different ontology inference engines for practical applications by translating ontology representation to any inference engine rule language.

Our future work is concentrated to experimental application of the provided logical model using the first-order theorem prover Gandalf [Tammet 1997].

Acknowledgements

This research was partially funded by Estonian Research Foundation by the Grant no 5766.

References

[Assadi 1999] Assadi H., Construction of a regional ontology from text and its use within documentary system, N. Guarino (Ed), Formal Ontology in Information Systems, Proc. Of FOIS-98, Trento, Italy, 1999, pp 236-249

[Baader et al 2002] Baader et al., (Eds), Desription Logic Handbook: Theory, Implementation and Applications, Cambridge University Press, 2002

[Decker et al 1999] Decker S., Erdmann M., Fensel D., and Studer R., Ontobroker: Ontology Based Access to Distributed and Semi-Structured Information, Meersman R. et al. (eds.): Semantic Issues in Multimedia Systems. Proceedings of DS-8. Kluwer Academic Publisher, Boston, 1999, 351-369.

[Fernandez-Lopez and Gomez-Perez 2002] Fernandez-Lopez M. and Gomez-Perez A., Overview and Analysis of Methodologies for Building Ontologies, The Knowledge Engineering Review, Vol. 17:2, 129-156, Cambridge University Press, 2002

[Gangemi et al 1999] Gangemi A, et al, An Overview of the ONIONS Project, Data and Knowledge Engineering, 31, 1999

[Ganter and Wille 1999] Ganter B. and Wille R., Formal Concept Analysis, Mathematical Foundations, Springer, 1999.

[Godin 1991] Godin R., Missaoui R., and Alaoui H., Learning Algorithms Using a Galois Lattice Structure, Proc. of the Third Int. Conference on Tools for Artificial Intelligence, IEEE Computer Society Press, CA, 1991, pp. 22-29.

[Grüninger and Fox 1995] Grüninger M. and Fox M. S., Methodology for Design and Evaluation of Ontologies, Workshop on Basic Ontological Issues in Knowledge Sharing, IJCAI-95, Montreal (available at www.eil.utoronto.ca)

[Haav 2003] Haav H-M., Learning Ontologies for Domain-specific Information Retrieval, W. Abramowicz (Ed), Knowledge-Based Information Retrieval and Filtering from the Web, Kluwer Academic Publishers, 2003, ch 15, pp 285-300

[Hobbs 1993] Hobbs J., The Generic Information Extraction System, Proceedings of the 5th Message Understanding Conference (MUC-5), Morgan-Kaufmann, 1993

[Hofmann 1999] Hofmann T., The Cluster-Abstraction Model: Unsupervised Learning of Topic Hierarchies from Text Data, Proceedings of 16th International Conference on Artificial Intelligence (IJCAI-99), Stockholm, Sweden, 1999, pp 682-587

[Jess] Jess Home Page http://herzberg.ca.sandia.gov/jess/

[Kifer et al 1995] Kifer M., Lausen G., and Wu J., Logical Foundations of Object-Oriented and Frame-Based Languages. Journal of ACM, 1995, 42(4), pp 741-843

[Kuznetsov and Objedkov 2001] Kuznetsov S. O. and Objedkov S. A., Comparing performance of Algorithms for Generating Concept Lattices, In: Proceedings of Internatinal Workshop on Concept Lattices-based KDD, ICCS´2001

[Maedche 2002] Maedche A., Ontology Learning for the Semantic Web, Kluwer Academic Publishers, 2002

[Maedche and Staab 2000] Maedche, A. and Staab, S., Discovering Conceptual Relations from Text, Proceedings of the 14th European Conference on Artificial Intelligence, IOS Press, Amsterdam, 2000

[Manning and Schuetze 1999] Manning C. and Schuetze H., Foundations of Statistical Natural Langauge Processing. MIT Press, Cambridge, 1999

[Sowa 2000] Sowa J. F., Knowledge Representation, Logical, Philosophical, and Computational Foundations, Brooks/Cole Thomson Learning, 2000.

[Stumme and Maedche 2001] Stumme G. and Maedche A., FCA-Merge: Buttom-up Merging of Ontologies, Proceedings of the 17th International Joint Conference on Artificial Intelligence, Seattle, USA, Morgen Kaufmann

[Tammet 1997] Tammet T., Gandalf, Reference Manual, University of Göteborg, Sweden, October 1997

[Tempich and Volz 2003] Tempich C. and Volz R., Towards a benchmark for Semantic Web reasoners-an analysis of the DAML ontology library, Sure Y (ed) Proceedings of Workshop of Evaluation of Ontology-based Tools (EON 2003) at 2nd Int. Semantic Web Conference (ISWC 2003), USA, 2003

SCIENTIFIC PAPERS UNIVERSITY OF LATVIA. 2004. VOLUME 672, Datbases and Information Systems
451–466 P.

Evaluating the Quality of Web-Based Ontology Building Methods: A Framework and a Case Study*

Sari Hakkarainen, Lillian Hella, Stine Tuxen, Guttorm Sindre

Norwegian University of Science and Technology
Sem Sælands vei 7-9, NO-7491 Trondheim, Norway
{sari, hella, stinemt, guttorms}@idi.ntnu.no

Abstract. Ontology is the core component in semantic Web applications. The employment of an ontology building method affects the quality of ontology and the applicability of ontology language. An evaluation approach for ontology building guidelines is presented in this paper. The evaluation is based on an existing classification scheme as introduced in a semiotic framework for evaluating the quality of conceptual models. A sample of ontology building method guidelines is analysed in general and evaluated comparatively in a small case study at an oil company in particular. Directions for further refinement of ontology building methods are discussed.

Keywords: semantic web, ontology building methods, quality evaluation

1. Introduction

The vision for the next generation web is the *semantic Web* [1], where information is accompanied by metadata about its interpretation, so that more intelligent and more accessible information-based services can be provided. The core components in the semantic Web and its applications will be ontologies. An ontology can be seen as an explicit representation of a shared conceptualization [2] that is formal [3], and will thus encode the semantic knowledge enabling sophisticated information services. The quality of a semantic Web application will be highly dependent on the quality of its underlying ontology. The quality of the underlying ontology will again depend on factors such as 1) the appropriateness of the language used to represent the ontology and 2) the quality of the engineering environment, including tool support and method guidelines, as provided for creating the ontology by means of that language.

There are also situated factors, such as the complexity of the specific task at hand and the competence of the persons involved. With a small number of developers the need for rigid method guidelines may be smaller than for larger projects. Similarly, with highly skilled modelling experts, the need for method guidelines may be smaller than for less experienced people. Method guidelines can thus be seen as an important means to make ontology creation possible for a wider range of developers, e.g., not only a few expert researchers in the ontology field but also companies wanting to develop semantic Web applications for internal or external use.

* This research has been partially supported by a research grant from Simula Research Lab.

Method guidelines can provide homogeneous instructions for creation of ontologies in a federated ontology engineering environment. However, the current situation is that while many ontology representation languages have been proposed, there is much less to find in terms of method guidelines for how to use these languages – especially for the newer Web-based ontology specification languages. Similarly, if there is little about method guidelines for Web ontology building, there is even less about evaluating the appropriateness of these method guidelines.

Hence, the objective is to inspect available method guidelines for semantic Web-based ontology specification languages. The approach is to adapt the method classification part of a model quality framework [4], use it to evaluate method guidelines and to validate the evaluation framework in a case study.

The outline is as follows. Section 2 describes related work. Section 3 describes seven categories in the classification framework. Section 4 classifies the selected method guidelines. Section 5 analyses their means to achieve quality goals in general and compared to the industrial case in particular. Finally, Section 6 concludes the paper and suggests directions for future work and for further refinement of ontology building methods.

2. Ontology Building Support and Evaluation Methods

Related work for this paper can be viewed from two angles: a) ontology representation languages and method guidelines for these, b) work on evaluating conceptual modelling approaches in general, i.e., languages, methodologies, method guidelines, and tools. The intersection between these two is fairly limited; the work on Web ontology languages has contained a little about evaluation, and the work on evaluating conceptual modelling approaches has concentrated on languages and mainstream approaches for systems analysis and design. However, the newer Web-based ontology languages are becoming mature enough to allow comparative analysis of their guidelines, given a suitable instrument.

During the last decade, a number of ontology representation languages have been proposed. The so-called traditional ontology specification languages include: CycL [5], Ontolingua [6], F-logic [7], CML [8], OCML [9], Telos [10], and LOOM [11]. There are Web standards that are relevant for ontology descriptions for semantic Web applications, such as HTML, XML and RDF. Finally, there are the newer Web ontology specification languages such as XOL [12] and SHOE [13], and those that are based on the layered architecture for the semantic Web, such as OIL [14], DAML+OIL [15], and OWL [6]. The latter group of the so-called semantic Web enabling languages (SWEL) for ontology building is in the focus of this study.

There exist several methodologies to guide the process of Web ontology building that vary both in their level of generality and granularity. Some of the methodologies describe an overall ontology development process yet not the ontology creation itself. Such methodologies are primarily intended to support the knowledge elicitation and management of the ontologies in a basically centralised environment:

> *Fernándes-Lopez et al. 1997* [17] proposes an evolving prototype methodology with six states as ontology life-cycle and includes activities related to project management and ontology management.

Sure and Studer 2002 [18] proposes an application driven ontology development process in five steps emphasizing the organisational value, integration possibilities and the cyclic nature of the development process.
Swartout et al. 1997 [19] proposes a top-down approach for deriving domain specific ontologies from common upper level ontologies and includes steps for requirements elicitation and for implementing the derived ontologies.
Uschold 1996 [20] proposes a general framework for the ontology building process consisting of four steps including quality criteria for ontology formalisation.

The above methodologies provide a life cycle in an overall ontology development process as analysed in [21, 22, and 20] but only a few user guidelines for carrying out the steps and for actually creating the ontology. In order to increase the number and the scale of practical applications of the semantic Web technologies, the developers need to be provided detailed instructions and general guidelines for the actual ontology creation. A limited selection of method guidelines were found for the Web ontology specification languages, which are at the foci of this study:

Knublauch et al., 2003 [24] present a tutorial containing method guidelines for making ontologies in the representation language OWL by means of the open source ontology editor Protégé
Denker, 2003 [23] present a user guide with method guidelines for making ontologies in the representation language DAML+OIL, again by means of Protégé.
Noy and McGuinness, 2001 [25] present method guidelines for making ontologies, called "Ontology Development 101". Unlike the other two, this method is independent of any specific representation language.

As for evaluation of ontology specification approaches, a comprehensive evaluation of representation languages was done in [26], covering all the languages mentioned above except OWL. The paper also evaluates some tools for ontology building: Ontolingua, WebOnto, WebODE, Protégé 2000, OntoEdit, and OilEd. Similarly, [27, 28] evaluate various ontology languages. These studies concentrate on evaluating the representation languages (and partly tools), not hands-on instructions or ontology building guidelines. Given the argumentation above, such studies are targeting the audience of highly skilled modelling experts rather than the wide spectrum of potential developers of semantic Web applications.

In the field of conceptual modelling there are, however, a number of frameworks suggested for evaluating modelling approaches in general. For instance, the Bunge-Wand-Weber ontology [29] has been used on several occasions as a basis for evaluating modelling techniques, e.g. NIAM [30] and UML [31], as well as ontology languages in [27]. The semiotic quality framework first proposed in [32] for the evaluation of conceptual models has later been extended for evaluation of modelling approaches and used for evaluating UML and RUP [33]. This framework was also the one used in the evaluation of ontology languages and tools in [26]. The framework suggested by [34] is particularly meant for requirements specifications, but is still fairly general. There are also more specialised quality evaluation frameworks, e.g. [35] for process models, and [36] for data / information models.

The framework used in [33] builds on an earlier framework [32]. This early version distinguished between three *quality categories* for conceptual models (syntactic, semantic, pragmatic) according to steps on the semiotic ladder [38]. The *quality goals* corresponding to the categories were syntactic correctness, semantic validity and completeness, and comprehension (pragmatic). The framework also took care to distinguish between goals and *means to reach the goals* (where, e.g., various types of method guidelines would be an example of the latter). In later extensions by Krogstie, more quality categories have been added, so that the entire semiotic ladder is included, e.g., *physical*, *empirical*, *syntactic*, *semantic*, *pragmatic*, *social*, and *organizational quality*. The framework has been adapted to evaluating specification languages by means of five categories [3], here adopted for evaluation of method guidelines as follows.

Domain appropriateness indicates whether the method guidelines address the problems of eliciting / representing relevant facts of the problem domain.

Participant knowledge appropriateness indicates whether the method corresponds to what the participants perceive as a natural way of working.

Knowledge externalization appropriateness indicates whether the method assists the participants in externalizing their knowledge.

Comprehensibility appropriateness indicates whether the participants are able to comprehend the method guidelines.

Technical actor interpretation appropriateness indicates whether the method guidelines lend themselves to automated tool support or assist in support for reasoning.

Since our evaluation is based on the method classification part of the framework of [4], it is most closely related to previous work using that same framework, and especially the evaluation of ontology languages and tools in [26]. In this paper the framework is used for evaluating something different, namely *method guidelines* for ontology building. Moreover, an interesting question is to which extent it is suitable for this new evaluation task, so customizations to the framework are suggested in order to improve its relevance for evaluating method guidelines in general, and method guidelines for ontology building in particular.

3. Criteria for Seven Method Guideline Categories

As argued in the introduction above, the developers typically need instructions and guidelines for ontology creation in order to support the learning and co-operative deployment of the semantic Web enabling languages in practice. [4] describes a methodology classification framework consisting of seven semiotic categories of modelling methodologies. We adapt the categories for classification of the ontology building method guidelines. The principle modification here is that the concept of application system (as the end product of the development process) is consequently replaced by ontology (as the end product of applying the method guidelines). In the following, the adapted criteria for each category are described briefly and the method guidelines are classified accordingly in the next section.

Weltanschauung describes the underlying philosophy or view to the world. For a method we may examine why the ontology construction is addressed in a particular way in a specific methodology. In the FRISCO report [38], three different views are described, namely the objectivistic, the constructivistic and the mentalistic view. *Objectivistic view* claims that reality exists independently of any observer. The relation between reality and the model is trivial or obvious. *Constructivistic view* claims that reality exists independently of any observer, but what each person possesses is a restricted mental model only. The relationship between reality and models of this reality are subject to negotiations among the community of observers and may be adapted from time to time. *Mentalistic view* claims that reality and the relationship to any model is totally dependent on the observer. We can only form mental constructions of our perceptions. In many cases, when categorizing a method, the Weltanschauung will not be stated directly, but should be derivable from the documentation.

Coverage in process concerns the method's ability to address 1) planning for changes, 2) single and co-operative development of ontology or aligned ontologies, which includes analysis, requirements specification, design, implementation and testing, 3) use and operations of ontologies, 4) maintaining and evolution of ontologies, and 5) management of planning, development, operations and maintenance of ontologies.

Coverage in product is described as the method concerns planning, development, usage and maintenance of and operate on 1) one single ontology, 2) a family of related ontologies, 3) a whole portfolio of ontologies in an organization, and 4) a totality of the goals, business process, people and technology used within the organization.

Reuse of product and process is important to avoid re-learning and recreation. A method may support reuse of ontologies as products or reuse of method as processes. There are six dimensions of reuse. Reuse by *motivation* answers the question - why is reuse done? Different rationale are for example productivity, timeliness, flexibility, quality, and risk management goals. Reuse by *substance*, answers the question – what is the essence of the items to be reused? A product is the set of deliverables that are produced during a project, such as models, documentation and test cases. Reusing a development or maintenance method is process reuse. Reuse by *development scope*, answers the question – what is the coverage of the form and the extent of reuse? The scope may be either external or internal to a project or organization. Reuse by *management mode*, answers the questions - how is reuse conducted? Reuse may be planned in advance with existing guidelines and procedures, or ad-hoc. Reuse by *technique* answers the question - how is reuse implemented? The reuse may be compositional and/or generative. Reuse by *intentions*, answers the question - what is the purpose of reused elements? There are different degrees of intention. The elements may be used as they are, slightly modified, used as a template or just used as an idea.

Stakeholder participation reflects the interests of different actors in the ontology building activity. The stakeholders may be categorized into those *responsible for developing* the method, those with *financial interest* and those who have *interest in its use*. Further, there are different forms of participation. *Direct* participation means every stakeholder has the opportunity to participate. *Indirect* participation

uses representatives, thus every stakeholder is represented through other representatives that are supposed to look after their interests.

Representation of product and process can be based on linguistic and non-linguistic data such as audio and video. Representation languages can be informal, semi-formal or formal, having a logical or executional semantics.

Maturity is characterized on different levels of completion. Some methodologies have been used for a long time; others are only described in theory and never tried out in practice. Several conditions influence maturity of a method, namely 1) if the method is fully described, 2) if the method lends itself for adaptation, navigation and development, 3) if the method is used and updated through practical applications, 4) if it is used by many organizations, and 5) if the method is altered based on experience and scientific study of its use.

4. Method Guidelines for Ontology Building - General Evaluation

Three method guidelines among the semantic Web-based ontology specification languages are categorized, namely *Knublauch et al., 2003* [24], which is based on OWL and Protégé, *Denker, 2003* [23] which is based on DAML+OIL and Protégé, and *Noy and McGuinness, 2001* [25] which is language independent yet uses Protégé in the examples. Protégé2000[1] is an open-source ontology editor developed at Stanford University and uses Java technology. All the method guidelines meet the selection criteria as supporting semantic Web applications as SWEL and assume RDF/XML notation rather than HTML or plain XML as the underlying Web standard. The studied method guidelines are shortly described and characterised in the sequel.

Knublauch et al., 2003 is a tutorial that was originally created for the 2nd International Semantic Web Conference. The ontology building method is based on OWL language and assumes Protégé as the ontology development tool. The ontology building process consists of seven iterative steps, namely determine scope, consider reuse, enumerate terms, define classes, define properties, create instances, and classify ontology.

Comment: The development activity requires some experience and foresight, communication between domain experts and developers, and a tool that is both comprehensible and powerful, including support for ontology evolution.

Denker*, *2003 is a user's guide of the DAML+OIL plug-in for Protégé2000. The ontology building method is based on DAML+OIL language and Protégé as the ontology development tool. The ontology building process consists of three basic steps; create a new ontology, load existing ontologies, save ontology. The creation of new ontology consists of five types of instructions; define classes, properties (slots), instances, restrictions, and Boolean combinations.

Comment: The method does not contain any explicit description of the development process. However, the sequence of the sections in the documentation gives an indication of how to create an ontology.

[1] Hereafter abbreviated Protégé as in http://protege.stanford.edu/

Noy and McGuinness, 2001 is a guide to building ontologies, called Ontology Development 101. The ontology building method is language and ontology development tool independent yet it uses Protégé in the examples. The ontology building process consists of seven iterative steps, namely determine the domain and scope of the ontology, consider reusing existing ontologies, enumerate important terms in the ontology, define the classes and the class hierarchy, define the properties of classes – slots, define the facets of the slots, and create instances.

Comment: The methodology provides three fundamental rules, for making development decisions, namely that 1) there is no single correct way to model a domain, that 2) ontology development is necessarily an iterative process, and that 3) concepts in the ontology should be close to objects, physical or logical, and to relationships in the domain of interest.

The classification of the selected method guidelines into the semiotic categories of [4] is summarized in Table 1. The columns of the table are the classification criteria as described above. The rows are the selected method guidelines. The intersection, i.e. the cells in the table, describes how the method meets the criteria. The studied method guidelines are characterised as follows.

Weltanschauung is similar in the studied methods. *(Knublauch et al., 2003)* is based on *constructivistic* worldview. The first step in the development method is to determine the scope. By doing that, the domain that is to be covered in the ontology will be explicitly stated. Further, the method states that communication between domain experts and developers is necessary. *(Denker, 2003)* is based on *undefined* worldview. The method does not explicitly state its worldview and it is not possible to implicitly deduce the worldview. However, this does not indicate that the guidelines have different worldview compared to the others. It merely indicates that the guideline is lacking information, which should be explicit or at least implicitly provided. The method does not define the term ontology, and it does not describe why an ontology is needed. *(Noy and McGuinness, 2001)* is based on *constructivistic* worldview. It presents a list of different reasons for creating an ontology, e.g. to make domain assumptions explicit. The method argues that an explicit specification is useful for new users. Thus, there is need for explanation, where the relation between the domain and the model is not obvious.

Coverage in process varies clearly between the methods. *(Knublauch et al., 2003)* covers seven iterative steps. It has a detailed yet unstructured and incomplete description of ontology development. The first three steps: determine scope, consider reuse and enumerate terms, are just mentioned. It describes the evolution and use of Protégé. The tool guidance does not follow the steps in the building process, but is presented rather ad hoc. There are no explicit procedures to prepare for changes. The process is described as iterative, which indicates the method awareness of and the need for modification. *(Denker, 2003)* covers three plus five steps. It has an unstructured and incomplete description of ontology development. The method contains no explicit description of the development process yet the sequence of the sections in the documentation indicates how to proceed in order to create an ontology. A detailed yet incomplete, description of how to create a DAML+OIL ontology with Protégé is provided. The importance of reuse is not covered and it does not describe how to plan for changes. It describes the

evolution and use of Protégé. It links to the syntax of DAML+OIL when its concepts in the development are described. Further the coherence between the development tool and the ontology language is considered important, i.e., resolving differences between the concepts of DAML+OIL and the representation in Protégé. There are explicit rules, e.g. that DAML+OIL properties are mapped to Slots in Protégé. *(Noy and McGuinness, 2001)* covers seven iterative steps, each of which is described in detail. It has good coverage in process. For example, step 1 (determine the domain and scope) is illustrated in different domains and the competency questions technique is suggested as a method to determine the scope. Reuse is considered, but there is no plan for changes. The actual implementation of an ontology is not covered. The method is an initial guide to help creating a single new ontology. It provides three fundamental rules in ontology design in order to make decisions. The process steps are covered in sufficient detail. For example, there are several guidelines for developing a class hierarchy. This feature provides participants a checklist to avoid mistakes such as creating cycles in a class hierarchy.

Coverage in product is medium (cover a single ontology) in both *(Denker, 2003)* and *(Noy and McGuinness, 2001)*. *(Knublauch et al., 2003)* includes an example scenario that describes the use of ontologies in relation to agents with reasoning mechanisms. It has high coverage in product. Protégé is described as a toolset for constructing ontologies that is scalable to very large knowledge bases and enables embedding of standalone applications in the Protégé knowledge environment. It does not describe the relationship between heterogeneous ontologies, nor the requirements the tool should fulfill prior to use in larger context. It refers to yet do not explain, description logics. *(Denker, 2003)* describes situations where the user would like to import concepts created in another ontology. The method does not allow references to resources located in another ontology except for four explicitly stated URIs[2]. The method covers single ontology. *(Noy and McGuinness, 2001)* regards an ontology as a model of reality and the concepts in the ontology must reflect this reality. It mentions projects built with ontologies, and ontologies developed for specific domains and existing broad general-purpose ontologies. Reuse is considered important if the ontology owners need to interact with other applications that have committed to particular ontologies or controlled vocabularies. Thus, there is awareness of the possible integration to other ontologies and applications. Further, translating an ontology from one formalism to another is not considered a difficult task. Instructions for this are not provided.

Reuse of product and process varies between the methods. *(Knublauch et al., 2003)* consider reuse partially in the ontology building activity. The *development scope* and *technical* prerequisite of reuse are covered. It does not describe why, when or how to consider reuse. It does not provide examples of how reuse is carried out in practice. It describes how to import existing OWL files that are developed with another tool or developed with some previous version of Protégé. It also lists formats from which ontologies may be read (imported), written to (exported) or inter-converted (transformed) between. *(Denker, 2003)* only considers technical

[2] http://www.daml.org/2001/03/daml+oil#, http://www.w3.org/1999/02/22-rdf-syntax-ns#, http://www.w3.org/2000/01/rdf-schema#, and http://www.w3.org/2000/10/ XMLSchema#

aspect of reuse. It explains how to import existing DAML+OIL files that are developed with another tool or developed with a previous version of Protégé. The process is described using images that guide the participants. However, the support tool, i.e. the plug-in only reads DAML+OIL ontologies and only allows such files to be manipulated and saved. *(Noy and McGuinness, 2001)* covers reuse in step 2: "consider reusing existing ontologies". Reusing existing ontologies is a requirement if the system needs to interact with applications that have already committed to some ontologies. Reuse is not fully covered yet references to available libraries of ontologies are given.

Table 1. Classification of method guidelines

	Weltan-schauung	**Coverage in process**	**Coverage in product**	**Reuse of product and process**	**Stakeholder participation**	**Representation of product and process**	**Maturity**
OWL- Protégé tutorial	Constructivistic	Iterative 7 steps detailed unstructured incomplete	Scenario incomplete	Step 2 How to Import existing files Lack of description	Created by Protégé team User support through mailing lists External community contributions	Informal Good graphical representation	Protégé is a well known tool, while OWL the newest ontology language
Tutorial: DAML+OIL with Protégé	Undefined	Description of Protégé and how to use the development tool	A single ontology	Only import of DAML+OIL files	Author unknown, link from DAML web site Refers to web site of Protégé	Informal Graphical description of how to use Protégé	Does not intend to be complete DAML-OIL is well examined
Ontology Development 101	Constructivistic	Iterative 7 steps detailed, examples of how to fulfil the steps Implementation and reuse is not covered.	Assumes that no relevant ontologies exist Claims that it is easy to translate ontologies, but does not provide any further info.	Step 2 in the development process Reference to available libraries of ontologies	Created by a member of the Protégé team and an editor of OWL. Comprehensible for both experienced and inexperienced users.	Informal Good graphical representation	Published in 2001. Referenced from many sites, examined by many readers.

Stakeholder participation discriminates the methods. *(Knublauch et al., 2003)* is developed by members of the Protégé team at Stanford University School of Medicine. The method assumes use of Protégé and provides a number of screenshots from the development tool. The tutorial is comprehensible for inexperienced stakeholders with development or financial interests and supports the interests of novice user participants. Since it is written by those *responsible for developing* the tool, the guide has a deep and detailed description of practical use. Several members of the user community, i.e. those who have *interest in its use*, have contributed to the method *indirectly* through material such as visualization systems, inference engines, means of accessing external data sources and user-interface features. *(Denker, 2003)* is available through the Artificial

Intelligence Center at SRI International, and is linked through the DAML homepage. The physical editor(s) author(s) are unknown other than the contact person regarding the plug-in and the user guide. In *(Noy and McGuinness, 2001)* one co- author is a member of the Protégé team and the other is co-editor of the Web Ontology Language (OWL). The method guideline provides introduction to ontologies and describes why they are necessary. The method is suitable for experienced as well as novice participants since it mainly uses informal languages, yet provides comprehensive descriptions.

Representation of product and process is only partially covered in all the methods. *(Knublauch et al., 2003)* is based on OWL and Protégé and the representations are influenced by these notations. It is mostly *informal,* written in natural language yet presents a narrow description of the Semantic Web and ontologies. On the *visual* part it has a multitude of screenshots that explain and make the semi-structured tool concepts and the *formal* language elements comprehensible. The development process is covered in a graphical representation yet not explained. In overall, the method is mostly *informal* and provides feasible graphical representation. *(Denker, 2003)* is influenced by the representations of DAML+OIL and Protégé. The document is basically written in natural language on top of screenshots that explain the ontology building method with Protégé. The user participant does not need to be aware of the underlying syntax of the ontology language. The document is accessed through links to the different sections that are to be opened/printed separately. The overall language and layout of the methodology are informal. *(Noy and McGuinness, 2001)* makes no explicit reference to any specific ontology language. It is written in natural language, with only a few logical or executable statements. The language is informal and the method offers adequate description of each concept presented. There are illustrations based on screenshots from Protégé that support comprehensibility. A semi-structured scenario is given and used as a reference throughout the guideline.

Maturity is covered on a medium level in all the methods. *(Knublauch et al., 2003)* is based on OWL, the newest contribution in this field. The language itself has hardly been examined yet. However, guidance for OWL modeling benefits from experiences with guidelines for Protégé, RDF and OIL. The plug-in that is used in Protégé is also new, but the core Protégé is well-examined. The method covers the latest release, and is up-to-date in both regarding the language and the tool. The method is not complete, since not all the steps in the development process are fully described. *(Denker, 2003)* is based on DAML+OIL as ontology language, released in December 2000. It has been subject for evaluation. Protégé is used by a large community and is a well-examined system. The method is not complete. However, the method guideline describes the uncovered or unimplemented functionalities. *(Noy and McGuinness, 2001)* was published in March 2001, and is older than the other two method guidelines. It is still valid when using ontology languages developed after the methodology was published, e.g. OWL. Many researchers in the field reference the method guideline, many readers examine it, and acknowledged Web sites such as the Protégé Web site provide hyperlinks to it. The method does not claim it has been tried out in practice, but by searching on the Web, several projects that use the method can be located. However, it has not been updated in response to such experiences.

In summary, the discriminating classification criteria between the studied method guidelines are in 1) *weltanschauung*, where the world view is explicit (constructivistic) in two of the method guidelines and undefined for *(Denker, 2003),* in 2) *coverage in process*, where none of the guidelines are fully complete, but *(Denker, 2003)* is the least complete, in 3) *reuse of product and process*, where *(Denker, 2003)* only mentions the support of import functionality, whereas the other two include reuse as one step in the building process, where *(Noy and McGuinness, 2001)* provides more description and functionality, and in 4) *maturity*, where *(Noy and McGuinness, 2001)* is the most mature. None of the method guidelines are even close to complete concerning *coverage in product* whereas all of them cover *representation of product and process* on a good or medium level.

4 Method Guidelines for Ontology Building – the edi Case

The case study is based on edi (engaging, dynamic innovation) which is a system developed by a student project group. edi is intended to support exchange of business ideas between the employees within a large Norwegian company, which is an integrated oil and gas company with business operations in 25 countries. At the end of 2002, there were over 15 000 employees in the company. Consequently, the amount of information and knowledge provided by the employees is rapidly increasing. There is an increasing need for more effective information retrieval and efficient sharing of knowledge. edi intended as idea management tool and a motivator for elicitation and generation ideas, as well as for enabling the employees to focus on the relevant aspects of their activities.

The overall approach for the edi system is to create a connection for communication and knowledge sharing between employees from different business areas as well as domain experts and department managers. The current plan is to utilize semantic Web and Web service technology for that purpose. Ontologies will play a crucial part in the edi system, both in supporting common access to information and enabling implementation of Web and ontology-based search. The participants will be experts on ontology building, on enterprise processes, on creativity and on processes that support creativity, all of which possess different qualities, modeling skills and domain knowledge.

edi requirements The status is that the functional requirements for edi have been analyzed. However, before the system can be developed a thorough analysis need to be conducted, and a decision about the purpose of the ontology has to be made. Information about the domain plays an important role in this process and it can be gathered in various ways. Unavoidably, there will be many different participants involved in such a process; for instance end users such as possible idea contributors and people in the edi network for evaluating proposed ideas. This is analogous to software development in general [27], hence starting with ontology requirements analysis. The requirements specification should describe what the ontology must support, sketching the scope of the ontology application and identifying valuable knowledge sources. Oil industry is a business in constant change, and the large international coverage of the company makes the changes

even more complex. edi needs to have high durability, be adaptable to changes in the environment, be maintainable and have high reliability in order to secure the investment. Thus, a careful analysis needs to be made early in the process, which places elaborate requirements on the ontology development environment.

Quality-based requirements An ontology should be built in a way that supports automatic reasoning and provides a basis for high quality Web-based information services. The underlying assumption is that a high quality engineering process assures high quality end product. The quality of ontology building process depends on the environmental circumstances under which the ontology is used. Further, a model is expected to have high degree of quality if it is developed according to its specification. Similarly, a method guideline is expected to have high quality degree if it describes a complete set of steps and instructions for how to arrive at a model, which is valid with respect to the language(s), it supports.

Table 2. Classification of method guidelines according to edi requirements

	Weltan-schauung	Coverage in process	Coverage in product	Reuse of product and process	Stakeholder participation	Repr. of product and process	Maturity
Edi -req. for methodology	Constructivistic world view – however this is not a crucial requirement	Extensively covered, with illustrations	Planning for a single ontology	Important, must be integrated in the process, Examples should be provided	Who created the methodology? What are the target groups?	Informal language Illustrations	Well examined
Offered by	M2 and M3 (undefined for M1)	M3, and partly M2	M1, M2 and M3	M3, and partly M2	M2, and partly M3	All, but prioritised list: M3, M2, M1	M3

As placed in the classification framework, the key criteria that are candidates for meeting edi requirements with high utility are *coverage in process*, *coverage in product*, *reuse of product and process*, and *representation of product and process*. In table 2, we have summarized which of the studied ontology building methods that meet the situated, quality-based requirements for the edi system. In the table, M1 refers to *(Denker, 2003)*, M2 refers to *(Knublauch et al., 2003)*, and M3 refers to *(Noy and McGuinness, 2001)*. The values in the first row of the table translate the edi requirements as categorized according to the evaluation criteria, whereas the values in the second row are based on the observations in the above section as contrasted to the above edi requirements. The values of the bottom row, i.e. the quality characteristics of the studied method guidelines in table 1 compared with the corresponding edi requirements in table 2, are explained in the sequel.

Weltanschauung - ontology building method for edi should be based on a constructivistic view. The end users may have different models of the reality depending on for example, their geographical location or the business area in which they are involved. - Both M2 and M3 meet this requirement, whereas the criterion is undefined for M1.

Coverage in process - ontology building method for edi should be extensively covered to support large development teams and heavily illustrated to support

inexperienced project participants. - Both M2 and M3 meet this requirement; M2 partially, whereas it is not well covered by M1.

Coverage in product - ontology building method for edi should cover a single ontology. - All the studied methods, M1, M2, and M3 guide creation of a complete single ontology.

Reuse of product and process - ontology building method for edi should provide feasible guidance including illustrative examples, and the procedures should be integrated into steps in the development process. - Both M2 and M3 meet this requirement; M3 partially, whereas it is not well covered by M1.

Stakeholder participation - ontology building method for edi should cover the participants' development and financial interests of the involved creators of the method as well as the low experience of its user group participants. Both M2 and M3 meet this requirement, M2 partially, whereas it is not covered at all or unknown by M1.

Representation of product and process - ontology building method for edi should cover informal (natural language) representation and rich illustration. Independent of the method, the language will cover the required level of formality in the product to support automated reasoning. - Each of the studied methods, M1, M2, and M3 uses both natural language and rich illustrations to support novice participants.

Maturity - ontology building method for edi should be widely adopted and well-examined in order to support evolution, co-operation and management of the ontology. - Relative to the other methods, M3 covers best the maturity criteria.

In summary, out of the key requirements for edi, the discriminating criteria are *coverage in process*, and *reuse of product and process*. M3 meet the both criteria completely, and M2 partially, whereas M1 fail in both cases. All the guidelines support completely *coverage in product* as required for edi and support the *representation of product and process* in a range, where M3 and M2 meet the requirements completely, and M1 only partially. Out of the remaining categories of edi requirements M1 fail to meet any of them, M2 meet two completely and fail in one, whereas M3 meet two completely and one partially. Thus, the situated evaluation is in favor of M3 and will be selected to guide the edi ontology creation.

5 Concluding Remarks

An evaluation of three method guidelines for semantic Web ontology building was conducted. The evaluation was instrumentalized adapting the method classification part of a framework previously proposed for evaluating modelling languages, e.g. as used in [33] for evaluating UML. Evaluation of method guidelines was performed in two steps, one general evaluation, i.e. their applicability for building ontologies in general, and one particular, i.e. how appropriate are they for ontology development in a real world project - how applicable is the framework in practice. The main results are as follows.

The method classification part of the semiotic framework [4] has potential for evaluating method guidelines. Some adaptation was needed in the 1) re-definition of the various appropriateness-criteria, and in the 2) application of the categories defined in [4], here diverting from the mainstream applications of the semiotic quality framework, e.g. in [32, 33, 36].

The categorization according to Weltanschauung, i.e. the applied modelling worldview, was expected to be the same for all the method guidelines, but turned out to be discriminating as selection criteria in the case study. However, the Weltanschauung most probably is the same for the studied guidelines, since they support languages, which all are constructivistic; it was merely not derivable for one of the guidelines.

In both steps, in the general classification and in the evaluation against the situated requirements, the method "Ontology Development 101" [25] came out on top, since meeting most of the evaluation criteria. This was also the only method guideline, which is independent of any specific representation language and has the longest history.

Major weaknesses were identified for all the methods, as expected because of the current immaturity of the field of Web-based ontology construction. None of the method guidelines are complete concerning *coverage in product* whereas all of them cover *representation of product and process* fairly well.

The main contribution of this paper has been to try out an existing evaluation framework with other evaluation-objects than it has been used for previously. This experience suggest that, given the small adjustments, the framework intended for model evaluation is fully applicable in evaluation of method guidelines regardless if the classification is used for their selection, quality assurance, or engineering.

The concrete ranking of methods may be of limited use, as new ontology languages and method guidelines are developed and the existing languages evolve and some of them became more mature. Nevertheless, it can be useful in terms of guiding the current and future creators of such languages and their method guidelines. Drawing attention to the weakness of current proposals, these can be mended in future proposals, so that there will be higher quality languages and method guidelines to choose from in the future. The underlying assumption for our work is that high quality method guidelines may increase and widen the range and scalability of the semantic Web ontologies and applications.

There are several interesting topics for future work, such as supplementing the theoretical evaluations with empirical ones as larger scale semantic Web applications arise utilizing the empirical nature of [32], as well as evaluating more methods as they emerge. The experiences from the case study, which here was intended as validation tool, suggest that numerical values could be used even for the classification and thus qualify weighted selection such as the [39] PORE methodology. Further possibilities are in investigating the appropriateness of the formalisation quality criteria in the [20] Unified methodology as a complement to the semiotic quality framework in order to conduct evaluation of the process oriented methodological frameworks that were out of scope of this study.

References

[1] Berners-Lee T, Handler J, Lassila O. The Semantic Web. Scientific American, May 2001.
[2] Gruber T R. A Translation Approach to Portable Ontology Specifications. Knowledge Acquisition. 5(2), 1993.
[3] Uschold M, Gruninger M. Ontologies: Principles, methods and applications. Knowledge Engineering Review, 11(2). 1996.
[4] Krogstie J. Conceptual Modeling for Computerized Information System Support in Organizations, PhD Thesis 1995:87 NTH, Trondheim, 1995.
[5] Lenat D.B., Guha R.V.: Building large knowledge-based systems, representation and inference in the cyc project. Addison-Wesley, reading, Massachusetts. 1990.
[6] Farquhar A, Fikes R. and Rice J.: The Ontolingua Server: A Tool for Collaborative Ontology Construction. Technical Report KSL-96-26, Knowledge Systems Laboratory, Stanford, CA, 1996.
[7] Kifer M., Lausen G., Wu J.: Logical Foundations of Object-Oriented and Frame-Based Languages. Journal of the ACM 42(4): 741-843, 1995.
[8] Schreiber G., B.J. Wielinga, J.M. Akkermans, W. Van de Velde and A. Anjewierden : CML: The CommonKADS Conceptual Modelling Language, In Proceeding of the 8th European Knowledge Acquisition Workshop, Belgium, 1994.
[9] Motta E.: Reusable Components for Knowledge Models. IOS Press, Amsterdam, 1999.
[10] Mylopoulos J., Borgida A., Jarke M., Koubarakis M.: Telos -- a language for representing knowledge about information systems, In ACM Trans. Information Systems, 8(4), pp. 325--362, 1990.
[11] MacGregor, R. and Bates, R.: The LOOM knowledge representation language. Technical Report RS-87-188, Information Sciences Institute, University of Southern California, 1987
[12] Karp P., Chaudri V. and Thomere J.: XOL: An xml-based ontology exchange language. http://ecocyc.panbio.com/xol/xol.html, 1999.
[13] Heflin J., Hendler J., and Luke S.: SHOE: A Knowledge Representation Language for Internet Applications, Technical Report CS-TR-4078 (UMIACS TR-99-71), 1999.
[14] Fensel D., Horrocks, I., van Harmelen F., Decker S., Erdmann M., and Klein M.: OIL in a nutshell In: Knowledge Acquisition, Modelling, and Management, R. Dieng et al. (eds.), Proceedings of the European Knowledge Acquisition Conference (EKAW-2000), LNAI, Springer-Verlag, October 2000.
[15] Horrocks, I., Patel-Schneider, P. F., Harmelen, F. v. : Reviewing the Design of DAML+OIL: An Ontology Language for the Semantic Web. AAAI/IAAI 2002: 792-797
[16] McGuiness D.L., and van Harmelen F. OWL Web Ontology Language Overview, W3C Recommendation, February 10, 2004.
[17] Fernándes-López M, Gómez-Péres A, Juriso N. METHONTOLOGY: From Ontological Art Towards Ontological Engineering. Proceedings of AAAI-97 Spring Symposium on Ontological Engineering. Stanford University, 1997.
[18] Sure Y, Studer R. On-To-Knowledge Methodology – Final Version. Institute AIFB, University of Karlsruhe. September 26, 2002.
[19] Swartout, B.; Ramesh P.; Knight, K.; Russ, T. Toward Distributed Use of Large-Scale Ontologies. Symposium on Ontological Engineering of AAAI. Stanford (California). March 1997.
[20] Uschold M. Building Ontologies: Towards a Unified methodology. The 16th Annual Conference of the British Computer Society Specialist Group on Expert Systems. Cambridge. United Kingdom, 1996
[21] Corcho O., Fernández-López M., Gómez-Pérez A. (2003) Methodologies, tools and languages for building ontologies: where is their meeting point?, Data & Knowledge Engineering, 46:1 July 2003, pp 41 – 64.

[22] Fernández-López M. Overview Of Methodologies For Building Ontologies, Benjamins V.R., Chandrasekaran B., Gomez-Perez A., Guarino N., Uschold, M. (Eds.), Proceedings of the IJCAI-99 workshop on Ontologies and Problem-Solving Methods (KRR5) Stockholm, Sweden, August 2, 1999.
[23] Denker G. DAML+OIL Plug-in for Protége 2000 – User's Guide. SRI International AI Center Report 7/8/03, 2003.
[24] Knublauch H, Musen M A, Noy N F. Creating Semantic Web (OWL) Ontologies with Protégé. Tutorial at 2nd International Semantic Web Conference. Sanibel Island Florida USA. October 20-23, 2003.
[25] Noy N F, McGuinness D L. Ontology Development 101: A Guide to Creating Your First Ontology. Stanford Knowledge Systems Laboratory Technical Report KSL-01-05, 2001.
[26] Su X and Ilebrekke L. A comparative study of ontology languages and tools. Halpin T, Siau K, Krogstie J (Eds.) Proc. EMMSAD'02, Toronto, Canada, May 27-28, 2002.
[27] Davies I, Green P, Milton S, Rosemann M. Using Meta-Models for the Comparison of Ontologies. Proc. EMMSAD'03, Velden, Austria, June 16-17, 2003.
[28] Gómez-Péres A, Corcho O. Ontology Languages for the Semantic Web. IEEE Intelligent Systems, 54-60, 2002.
[29] Wand Y, Weber R. Mario Bunge's Ontology as a formal foundation for information systems concepts. In: Weingartner P and Dorn G (ed.): Studies on Mario Bunge's Treatise, Rodopi, Atlanta, 1990.
[30] Weber R, Zhang Y. An analytical evaluation of NIAM's grammar for conceptual schema diagrams. Information Systems Journal 6(2): 147-170, 1996.
[31] Opdahl A L and Henderson-Sellers B. Ontological evaluation of the UML using the Bunge-Wand-Weber model. Software and Systems Modelling (SoSyM) 1(1): 43-67. Springer, 2002.
[32] Lindland O I, Sindre G, Sølvberg A. Understanding Quality in Conceptual Modeling. IEEE Software, 11(2): 42-49, 1994.
[33] Krogstie J. Using a Semiotic Framework to Evaluate UML for the Development of Models of High Quality. Siau K and Halpin T (Eds.) Unified Modeling Language: Systems analysis, design, and development issues. IDEA Group Publishing, 2001.
[34] Pohl K. Three dimensions of requirements engineering: a framework and its applications. Information Systems 19(3): 243-258, 1994.
[35] Becker J, Rosemann M, von Uthmann C. Guidelines of Business Process Modeling. In W. Aalst, J. Desel, A. Oberweis (Eds.): Business Process Management: Models, Techniques and Empirical Studies. Berlin, 1999.
[36] Moody D, Shanks G, Darke P. Evaluating and Improving the Quality of Entity Relationship Models: Experiences in Research and Practice. Proc 17th International Conference on Conceptual Modelling (ER '98), Singapore, 1998.
[37] Schuette R. Architectures for evaluating the quality of information models – a meta and an object level comparison. In Proc. ER'99, Paris, France, 1999.
[38] Falkenberg E D, Hesse W, Lindgreen P, Nilsson B E, Han Oei J L, Rolland C, Stamper R K, Van Asssche F J M, Verrjin-Stuart A, Voss K. FRISCO - A Framework of Information Systems Concepts. IFIP WG 8.1 Technical Report. December 1997.
[39] N. Maiden, C. Ncube, "Acquiring COTS Software Selection Requirements", IEEE Software March/April 1998, pp 46-56.

INFORMATION SYSTEMS AND SECURITY

SCIENTIFIC PAPERS UNIVERSITY OF LATVIA. 2004. VOLUME 672, Datbases and Information Systems
469–480 P.

e-EDU – An information system for e-learning services

Tarmo Robal, Ahto Kalja

Department of Computer Engineering at Tallinn University of Technology
Raja 15, 12618 Tallinn, Estonia
tarmo@pld.ttu.ee, ahto@cs.ioc.ee

Abstract. There have been many different educational and learning support systems, some of them have failed, and some have been more or less successful. The general practice among lecturers has been to develop their own systems to manage tasks, study results, materials. The e-EDU provides a general approach to solve these problems, taking into account the real needs of the lecturers of TUT. The e-EDU system proposes a new and localized approach as a service to enliven the daily studies at the university as well as offers means for distance learning.

Keywords. e-learning, distance learning, study systems, web applications, information systems, XML, RUP, service- oriented systems

1. In need for new approach

Over the years, universities have developed several different kinds of information systems, including systems that were supposed to support learning processes via providing materials, links and other information valuable during an educational course.

The majority of universities' information systems concentrate on the management of study process results, students and the subjects they have performed. These kinds of systems actually deal with the students' information at the beginning of the course, when students have to declare courses in the inception of a new semester. The next phase, when these systems deal actively with students' information is the end of the term, when the results need to be entered into the system. As we see, usually there is no support service for the course itself. The general practice has been that each lecturer has his or her own system, how to manage the course: students, their tasks and results, course materials. It is rather a lecturer-central system than an online service.

With the WebCT, Lotus LearningSpace [1,2] and other systems alike, these problems can be revealed. However, in most cases they are rather difficult to use and do not satisfy the real needs of the academic personnel of a particular university, country, region or an educational system.

The e-EDU stands for an online learning support service being developed at the Department of Computer Engineering at Tallinn University of Technology (TUT). Its purpose is to provide a supporting service for students and lecturers. Two lecturers, who had similar systems on their own, invoked the e-EDU development. Today, the system, which grew out of these two early versions, is being used as an

active learning support system in seven subjects taught by the department. The subjects are Informatics I and II, Computers I and II, Digital Systems Design and Test, Testing of digital systems, Fault-tolerant systems. These courses are being taught to more than 200 students each semester. The system has been under development from the year 2002. The e-EDU takes into account the real needs of the lecturers of the department and in the nearest future will compete with WebCT as the basic learning system for distance learning. It's not yet another WebCT!

We have to take an advantage of tomorrow's technologies already today!

2. Access from everywhere

One of the advantages of the e-EDU system is that it is a web-based system with many interfaces. Therefore, it can be accessed from almost everywhere, regardless of whether you are using a PC, Workstation, PDA, or Laptop. The system is built up on the assumption of service-based architecture in a way that several interfaces are supported. For example, lecturers do not need to have any student information on paper, instead they have their PDA, Laptop or other portable device and can have an access to the IS via LAN or WiFi. The majority of premises of Tallinn University of Technology are covered with WiFi network, which is a good prerequisite for such a system development and exploitation.

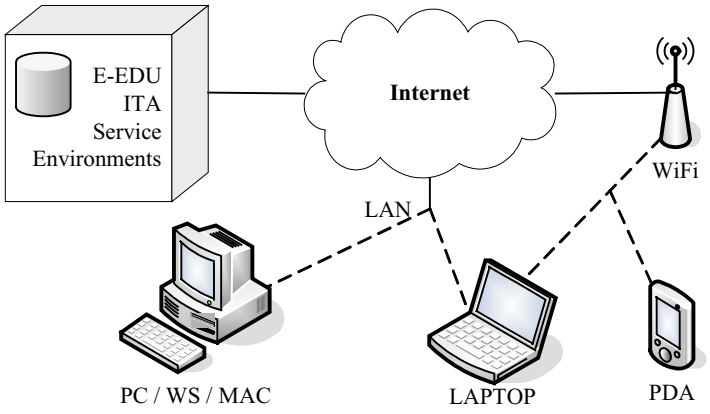


Figure 1. Ways to access the service.

3. System functionality

The functionality of the e-EDU system is based on the needs of students and the academic personnel. The main goal of the service is to decrease and mitigate the information processing of student data performed by the academic personnel. On the other hand, as technological potential improves, we need to consider today's and tomorrow's means for making the study process more effective.

As far as the system development is concerned, the concepts of web applications development discussed in the author's earlier paper: The Rational Unified Process with the "4+1" view model of Software Architecture – a Way for Modelling Web Applications [3], are being applied. The system's architectural design is described using the Unified Modelling Language, the graphical user interface as mock-up pictures.

The e-EDU system has three major views, according to the users, that are implemented via different web services. The views are as follows:

- The students' view, as a web application edu.pld.ttu.ee;
- The teachers' view, as a part of the department's intranet application ITA;
- The administrator's view, as a part of the ITA intranet application;

The functionality provided by the e-EDU can be grouped as follows:

- User authentication via a general authentication system or using the Estonian Citizen ID-card;
- Course registration management;
- Registered students management;
- Course material management;
- Assignments and students' results management;
- Communication between academic personnel and students;
- Tests generation and crediting;
- General course management – administrators only;
- Archived data management – administrators only;
- System back-up management – administrators only.

As we are dealing with a system that is rather laborious and risky to implement fully, system elaboration has been divided into development cycles by functional modules according to the demand for certain functionality. The development is based on the RUP technology. The system development started off with the most critical part: assignments and results management. The next development cycle added the management system for course materials. Today, a third cycle of development of this module has been finished. In the nearest future, other functional modules will be developed in the following order: announcements subsystem, test generation and evaluation subsystem, communication subsystem for data exchange with other e-learning systems. Further development of already existing modules will continue, in order to add functionality step-by-step, improve the systems functional behaviour and to make the applications user-friendlier.

Both, the e-EDU web portal and the ITA intranet application are represented over the Internet using the HTTPS protocol with applied security certificates to attain more secure connection than the regular HTTP protocol would provide. Also, organizational measures are applied to enforce the security. As we see later in this article, the lecturer can perform the majority of activities with the course; however, there are operations, which can be carried into effect only by an administrator. System development and testing are rigorously separated from the operative "LIVE" system. As security descriptions are not the aim of this article, we will not discuss them in further detail.

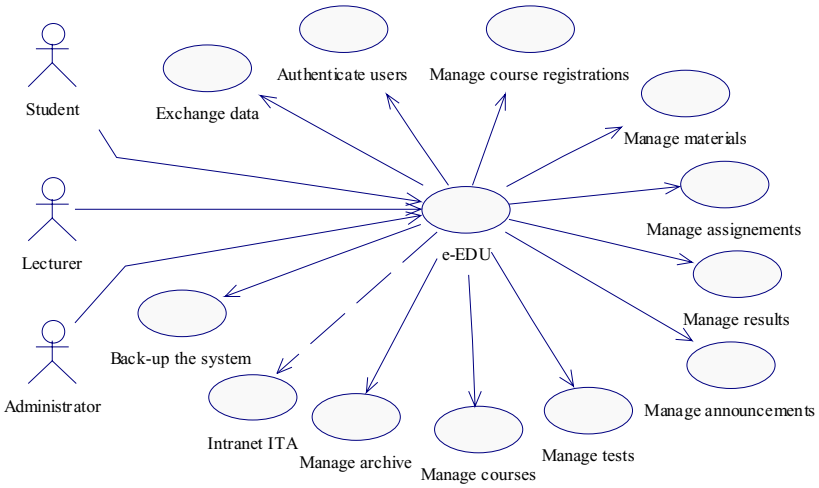


Figure 2. Main functional groups of the e-EDU system.

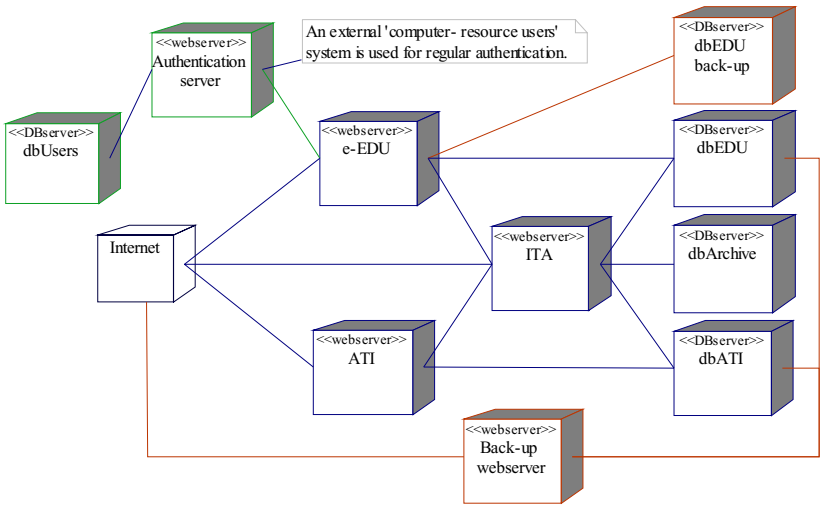


Figure 3. Architecture of the e-EDU system.

3.1. The students' view

The students' view is implemented as a standalone web portal edu.pld.ttu.ee accessible all over the world via Internet. The system has a built-in language support, so that users can select the language in which they prefer to interact with the system already at the stage of login. The default language is Estonian. To be able

to work in the web application, users first need to authenticate themselves, which can be done in three different ways:

- User enters his or her account information for the computer resources at The Computer Centre of TUT and can have an automatic access to the system;
- User enters his or her e-EDU username and password;
- User identifies himself or herself using the Estonian Citizen ID-Card. A Personal Key Infrastructure (PKI) [4] is used for identification.

After a successful authentication, users can access the functionality of the students' view of e-EDU. They can manage their personal data in the system, for instance change their e-mail address, customize the e-EDU application for their preferences (preferred system language, colours, remainder, whether they want to get the announcements also to their e-mail, etc), subscribe to subjects and of course, manage the courses they have registered for.

In order to be able to operate with a certain course, users are required to register for it. During the registration process they have to choose the lectures and the study group in which they will take part of the lessons. This allows the academic personnel to know the actual distribution of students of a particular course. On the other hand, students can freely choose another time, study group or lecturer to take a course with. After a successful registration, users can:

- Take their personal assignments and view the state of taken assignments;
- View the results of the assessments;
- View reports about their success compared to other course students;
- Surf the study materials, both, free and password protected materials;
- Take online tests;
- Check-in for a lesson online (computer classes only);
- Activate the remainder function, not to miss the deadlines for the assignments;
- Take part in the subject communication board;
- Read course announcements, descriptions (extended course information cards) and calendars.

Each student gets his or her own personal version of a task. After a successful registration procedure students are able to take their assignments. If there is more than one assignment in a course, lecturers have the possibility to set preconditions to assignments release. For example, in order to take *task 3*, student has to have *task 2* defended and its status set to "OK". This will unable students to perform tasks, which have prerequisites, and the acquiring of needed knowledge in the right order can be assured more accurately. At the same time, it acts as an external motivator – each task has its deadline and preconditions. Moreover, the length of the term is limited (at TUT it is 16 weeks), which leads to restricted time limitations. Each assignment can be viewed as a web page, printed out or saved as a PDF-file. The assignments are presented as personal forms: it has a student's name, task body, variable task part, task deadline, date of task taking and the current status of the assignment among with other information on it. Also, barcodes can be generated for the tasks. For example, students get their personal home tasks (for instance a blank form with student's name, task information and barcode on it), print them out with a

barcode on it and submit them to the lecturer. The barcode then makes the procedure of evaluating and results entry easier and more efficient.

The tests system acts like the assignments subsystem. If there are tests put up by the lecturer, and they are made available, students can take them. Tests can be either online web forms for filling in or in a printable format, labelled with student information and barcode.

General course description represents the extended information card of a subject. The course calendar shows the topics for an active study week, highlighting them among the rest of the records. Course materials are represented by their type, whether they are materials given during a lecture, materials needed in tutorials or labs. A search form is also provided.

Each course is provided with a communication board, where students and lecturers can exchange knowledge and discuss problems with each other. It acts more or less like a public forum, with the exception, that each course has its own independent section.

The *e-check-in* is a feature, which is only usable with a direct access to computer resources. It can be used in daily study process as well as with distance learning. For instance, lecturer and students of the distance learning have agreed to be online at the same time to hold an online lecture. In daily study the e-check-in feature allows to check the presence of students in a lecture without sending out a special list with names. Teachers can set the timeframe for e-check-in service or manually open the e-check-in for a period of time, when participants can sign their presence. The system has a built-in feature to allow signing only from certain address range (*i.e.* IP-addresses), if needed. For example, daily study e-check-in can take part only in the computer classes of TUT.

The functionality of the students' view is elaborated taking into account the needs of daily study students as well as the needs of those of distance learning. The central line is to keep everything as simple as possible, providing as much information as needed. In comparison with WebCT and other e-learning systems, e-EDU is more flexible, more automated, user-friendlier and which is most significant, it is in Estonian language. The latter is a key success factor in exploiting such a system, making it as user friendly as possible and at the same time escaping the confusion in users.

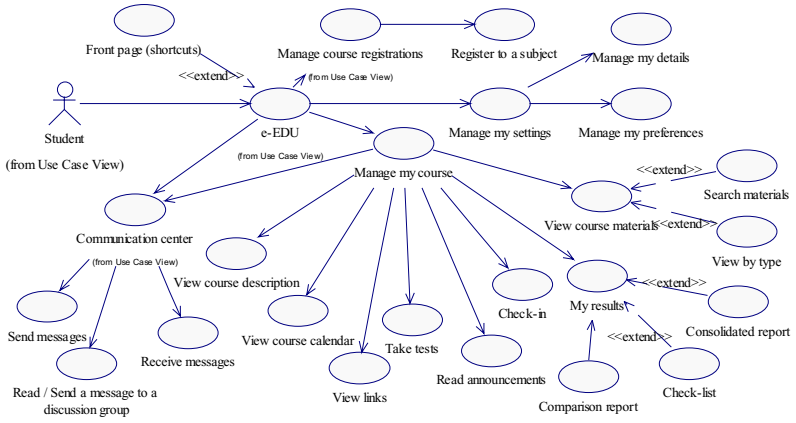


Figure 4. Functionality of the student's view.

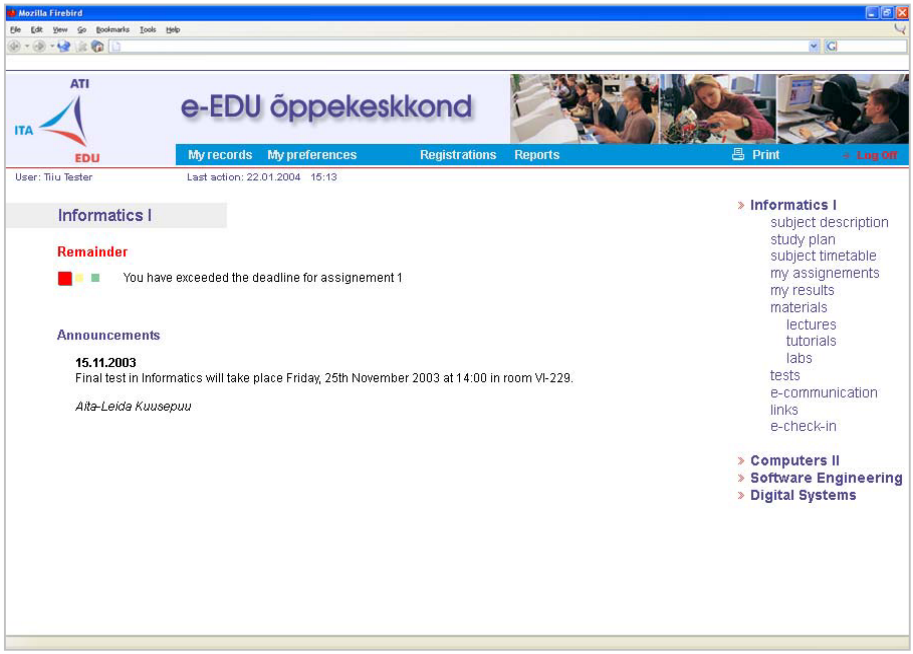


Figure 5. The student's view.

3.2. The teachers' view

The teachers' view is a part of the department's intranet called ITA. The section of ITA, which deals with the e-EDU system, has also a secondary interface to work with PDA-s. Through the system, lecturers can:

- Easily see students who attend their courses,
- Upload, delete and reassign course materials,
- Evaluate students' assignments,
- Carry out automatic assessment,
- Send announcements to the course participants,
- Compose and evaluate tests,
- Generate statistical reports,
- Manage examination information in the Examination Centre (Sessikeskus),
- Import and export data in CSV format (for example exchange data with WebCT, Excel, OpenOffice)
- Manage course descriptions, course calendar and schedule.

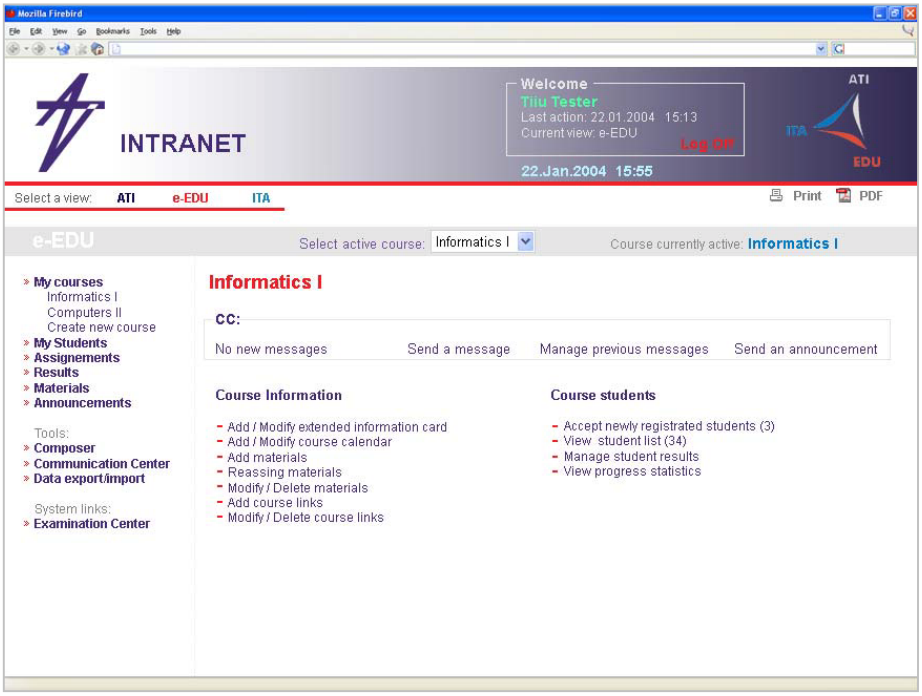


Figure 6. The teacher's view.

Although each course has its general description in the central registry of the university, each year minor modifications are being made to the course program and actual content. Therefore, lecturers are ordered to compose extended course content and schedule plans, called extended course information cards. The idea is to provide a description of a course, sufficient for students, so that they are able to make rational subject selections. On the other hand, extended course information cards keep lecturers "on the track" assuring that students attain the knowledge needed. In addition to extended cards, course calendar provides an overview of the course structure as a timeline. There are two different approaches to choose:

- Timeline as academic hours: describes how many academic hours are spent on each topic;
- Timeline as lecture topics: describes what topics will be discussed in a concrete lecture.

The timeline itself shows the logical order of covering the topics during the whole course.

The most significant part of the system is the assignments management. The subsystem uses XML-based documents description technology. Tasks are described as XML-based documents; a coherent style is assigned using XSLT-s. As a result, the assignment management system gives us the following benefits:

- Due to the use of XML and XSLT, tasks have a general coherent outlook, but may have a different style, if needed,
- Tasks can be represented in various forms: HTML, PDF, etc.

- The stationary task body and variable version information are kept separately until a student takes an assignment,
- The distribution system guarantees even allocation of different assignment versions,
- Each student gets his or her own personal version of an assignment, composed from the task body and variable task part, which can be easily stored in an archive,
- Teachers do not have to print out and distribute tasks; everything is organized by the assignment management service and is available in e-EDU.

The exploitation of XML-based document description technology also provides simple interfaces for communication with other systems.

The composition of assignments and tests is implemented also by a web application, where the lecturer has to make selections to provide the system with needed information (i.e. number of task variants) to produce the basic template. After the basic template has been generated, the task information can be entered into the database. By default the tasks are unavailable and lecturer has to give a special order to the system to make the tasks available. The task sub-system generates needed XML structures as well as makes necessary entries into the database.

Results management is common for both, assignments and tests. If a student has taken an assignment or a test, his or her name appears automatically in the task's results list. This way the system assures the integrity of data and implementation of the rules, decrementing the workload of academic personnel. Of course, there is a possibility to add tasks to students or predefined student groups as well. The default assessment types are:

- Defended and passed;
- In proceeding;
- Cancelled;
- Indefinite.

When a student takes a task, it is in an indefinite state, after the evaluation it may have other states. The final state is either *Defended and Passed* or *Cancelled*. With the cancellation, student has to take the task again. The system assures, that the version does not mach with the cancelled one.

Teachers are able to upload materials in any form they like; it does not matter whether the materials are HTML-files, PDF-files, PowerPoint slide presentations or other types of file. While uploading, they need to specify, whether the materials are for public use or are secured (accessible only through e-EDU). The files can also be classified as lecture, tutorial or lab materials, to provide a better overview of them.

The aim of statistical reports in the teachers' view is to provide a general overview of the progress of students, which is useful and needed information for a lecturer. It enables to adjust the course according to the success of students during the teaching process.

The teachers' view provides a lot of useful tools to make the teaching process easier, simpler and lessen the workload, but it can only provide the functionality built in it, which is rather general and may not correspond to all of the needs of a particular lecturer. However, the system is developed by the actual needs of the department's lecturers and therefore can answer the demands better than any other outsourced software.

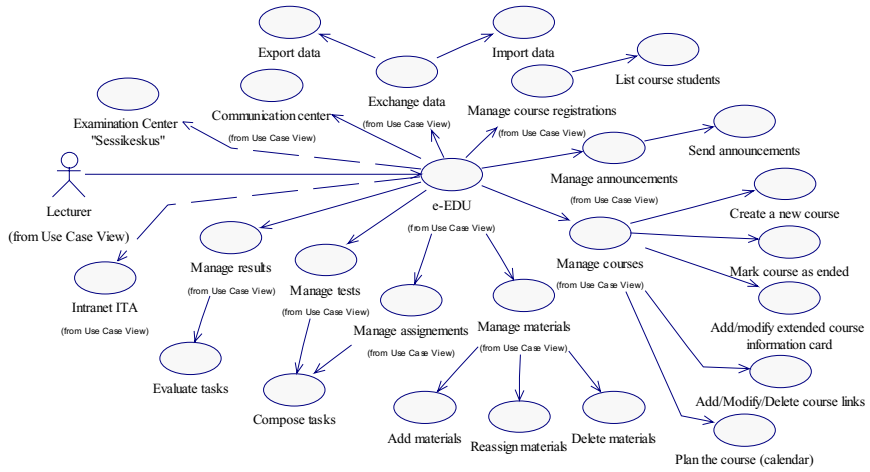


Figure 7. Functionality of the teachers' view.

3.3. The administrator's view

The role of an administrator in the e-EDU system is principally to act as the supervisor and technical assistant for the academic personnel as well as for the students. As there should be some kind of superintendence implemented over the two other user groups, the administrator is the only actor, who has the rights to set up new and erase old courses, add, delete and modify data concerned with users, both, students and academic personnel. Moreover, the administrator's tasks are not limited with simple course and user management. They can be extended to auditing certain actions taken by the academic personnel. For instance, administrators may have the right to audit and certify the extended course information cards and course calendars created and modified by the academic personnel.

It is evident that with this kind of systems, we are most probably interested in several types of statistics over the applications usage, or even more, there might be a need to get information about the applications usage for a concrete user. Obviously, this sort of information might be delicate and everyone should not have access to such a data.

One year after the course has ended, administrator transfers the course information into archive. After archiving, the data will be available only for administrators, who can excerpt reports, if needed. This approach frees the active server from huge data loads, on the other hand, provides a mechanism, where all the data remains obtainable.

Despite of the technological possibilities and regulations of the department, the role of the administrator should be as minimal as possible, incrementing the independence and responsibility of the academic personnel. Divided responsibility is more or less irresponsibility. As we have shown, the role of an administrator is mainly to function as a superintendent and plays a crucial role in such systems.

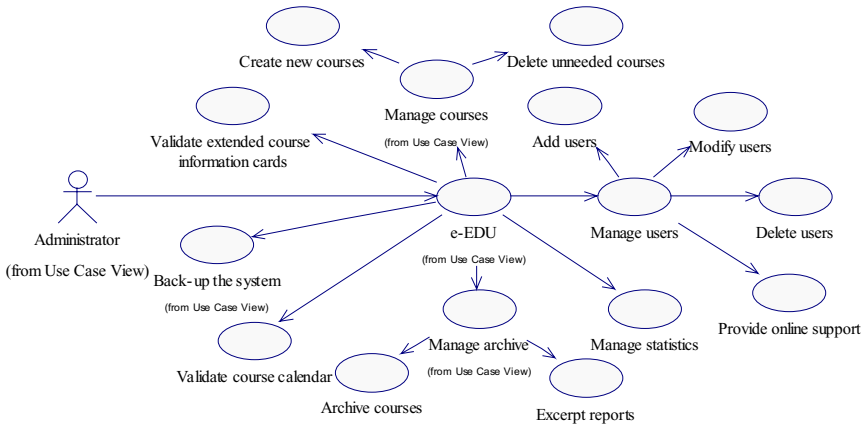


Figure 8. Functions of an administrator.

3.4. Overview of the physical implementation

The e-EDU system implementation involves many database and web servers as can be seen on Fig. 3. The system is implemented using mainly Apache web servers with PHP modules (currently version 4.x) added [5]. The database servers are implemented using the MySQL database engines (currently versions 3.23.x and 4.0.x) as the most popular open source and the fastest growing database systems in the industry [6]. MySQL has proved to be suitable for such systems, however presently there are some restrictions: there are no triggers, functions or possibilities to run sub-queries. These restrictions do not hold back the development, as there are other possibilities to implement the needed functionality, and hopefully these restrictions will be eliminated by future versions of MySQL.

The main line has been and will be to keep the system as simple as possible, using the available freeware and implementing the functionality mainly by PHP, Javascript and MySQL.

4. Future work

In the future, further development of the system will be continued, in order to add more functionality, make the system more efficient and more open for integration with other information systems, first and foremost with the department's information systems but also with the university's systems. It has been planned to develop a communication link between the e-EDU system and the study information system (OIS, http://ois.va.ttu.ee) of TUT. Moreover, the e-EDU can be easily integrated with the Career Service Web Portal (CSWP, http://www.ttu.ee/karjaar) of TUT. In the future it would enable the users of e-EDU to access the services offered by CSWP without registering for another username and maintaining another password.

In the next version we have planned to introduce the Estonian Digital Signature for the communication between lecturers and students to enable digital signing of documents. It is highly likely that the future also brings M-Services (Mobile Services) to e-EDU, allowing, for instance, students to access their data by just sending a SMS or accessing the portal using the WAP.

5. Conclusions

The rapid IT development has provided us with tools to improve our lives, our possibilities to teach and learn. Web applications are one of the many possible approaches. Because of their open nature, users usually do not need anything more than just a web browser and access to Internet - web based applications have become very popular, also in the field of education.

The e-EDU system provides a new and localized approach to enliven the daily studies at the university. In addition, it provides means for distance learning. As has been shown, the e-EDU system lessens workload of teachers, providing general means for tasks, materials and results management among with other tools. There is no need for every teacher to manage his or her own mini-systems in Excel, Access or in any other program. Students have a continuous overview of their progress and in case of errors made by a teacher, can quickly announce of it.

There have been many steps towards implementing today's technological means in providing education and e-education. Many of them have failed, whenever the systems have been too complex to manage or just from lack of interest by users. The e-EDU system is making its first steps and until now has been successful.

6. Acknowledgements

The project has partly being supported by the Estonian Scientific Foundation grant no. 5766. We would like to take this opportunity to thank all those people who have collaborated and supported this project: Margus Kruus, Elmet Orasson.

References

[1] WebCT.com, http://www.webct.com/ (03.12.2003)
[2] LearningSpace, http://www.lotus.com/products/learnspace.nsf/wdocs/homepage/ (03.12.2003)
[3] Robal, T., Viies, V., Kruus, M. The Rational Unified Process with the "4+1" View Model of Software Architecture – a Way for Modelling Web Applications. In.: Proceedings of the Fifth International Baltic Conference, BalticDB&IS 2002, Tallinn, June 3-6, 2002, Vol. 2., pp.119-132
[4] Kalja, A., Vallner, U. Public e-Service Projects in Estonia. In.: Proceedings of the Fifth International Baltic Conference, BalticDB&IS 2002, Tallinn, June 3-6, 2002, Vol. 2., pp. 143-154.
[5] PHP Hypertext Preprocessor, http://www.php.net/ (23.11.2003)
[6] MySQL, http://www.mysql.com/ (23.11.2003)

SCIENTIFIC PAPERS UNIVERSITY OF LATVIA. 2004. VOLUME 672, Datbases and Information Systems
481–496 P.

A Contribution to Web Digital Archive Integration from the Parliamentary Management System 'SIAP'

Carmen Costilla[1], M.ª José Rodríguez[2], Juan P. Palacios[2], José Cremades[3], Antonio Calleja[3] and Raúl Fernández[3]

[1] Titular Professor, *Information Systems and Databases* (SINBAD) Research Group
Dept. Ingeniería de Sistemas Telemáticos (DIT), ETS Ingenieros de Telecomunicación (ETSIT), Technical University of Madrid (UPM), www.sinbad.dit.upm.es, costilla@dit.upm.es

[2] Ph. Researchers at the SINBAD- DIT-ETSIT-UPM
www.sinbad.dit.upm.es, mjrodriguez@sinbad.dit.upm.es; jppalacios@sinbad.dit.upm.es

[3] Ph. Researchers at the SINBAD-DIT-ETSIT-UPM and
Founder Partners of CRC Information Technologies
(www.crcit.es), {jcremades, acalleja, rfernandez}@crcit.es

Abstract. Nowadays, millions of users choose the web as their main communication channel. As a consequence, everyday we find more activities (government, commerce, finance, digital libraries…) developed through the web. Archives are an example of this trend. They are being digitized and there is already an important initiative to standardize web access to them. Firstly, this paper introduces the *SIAP* system we have built; it has been running at the Asamblea of Madrid since 1999. Secondly, we present DAWIS-UPM[1] project (Digital Archive Web Information System) whose main goal is to design a web architecture providing integrated access to different digital archives. This architecture is based on mediators and wrappers making the virtual, flexible and dynamic 'on-the-fly' integration of digital archives possible. Finally, a semantic integration -based on ontologies, according to current international archival standards- is here proposed.

Keywords: Semantic Web, Ontologies, Web Information Systems (WIS), Integrated web architecture (Mediator & Wrappers), DCMI, ISAD(G), ISAAR(CPF), Digital Archives (DA), Parliamentary Information Systems (SIAP), e-government.

1. Introduction and Motivation

The web is a universal information space that every day freely incorporates new server sites that may contain very different Web Information Systems (**WIS**). Web data management is a relevant topic for governments, museums, libraries, businesses and other institutions. As the current web is inherently heterogeneous in data formats and data semantics, the semantic web is very necessary.

The web is anarchic with regard to the capabilities of linking semantic affinity information contents of a specific domain. However, the fundamental is that -on the

[1] DAWIS-UPM stands for **D**igital **A**rchives **W**eb **I**nformation **S**ystem, the TIC2002-04050-C02-02 national research project (2002-2005), funded by the Spanish Ministry of Science and Technology within Communications and Information Technologies National Programme, developed at the DIT-ETSIT-UPM.

web- these domains could have a universal scope, and this is one of the most important communications and computer science goals you can get. Today, the semantic web constitutes an important challenge for future web intelligence [7, 49]. Current research proceeds quickly in this direction.

Research into analyzing web sites faces two challenges due to the huge amount of on-line published data and the complex structures found in them [3]. We think it is absolutely essential to improve the virtual integration of heterogeneous and distributed **WIS** [46]. Focused on a specific domain, the web will allow the dissemination of unified and useful information to the user [27].

We have today new web technologies related to interoperable web, such as XML and web services [4]. However, web data integration has to improve considerably. **WIS** are being still integrated by hand –with tedious processes and *ad hoc* work- that produce highly vulnerable results. To cover current needs -when many different sources are offering a wide variety of semantics- it is not enough. Besides, the related software applications also present a high degree of diversity.

In this context, archives are a very interesting and representative WIS investigation area. From a technical point of view, archives hold huge amounts of documentary information, where the multimedia, the diversity of material and format aspects is the summit of all its splendour. The (semi-) automatic integration of many web Digital Archives (**DA**) permits user easily access to global information.

This paper describes some aspects of web DA integration, thanks to the experience acquired from our recent lines of research. A just finished project is due to a Parliamentary Integrated Management System, -***SIAP***[2]-, that we have built. ***SIAP*** was funded by the Parliament of Madrid, whose first product (called SGP) has been running successfully at the *Asamblea of Madrid* since 1999 (www.crcit.es/ SIAP). Nowadays, our ongoing research is focused on heterogeneous web data source integration and semantic web, applied to web DA virtual integration.

We have investigated DA conceptual modeling in order to define a virtual web integrated architecture through a unified semantic **mediator layer** (made up of ontologies, mappings and data repositories) and **wrappers** (coupling heterogeneity between data sources, using XML). We are starting this current investigation, and our final (and future) idea is the implementation of our results as different web service levels (Java libraries) of a web DA integration architecture.

We think that building a web DA integrated architecture of distributed heterogeneous data integration, with dispersed components and services, running very different web functions is a paradigmatic application [44]. As well as the services, the overall functionality depends on the ability of the specification and implementation of semantics that each web query requires.

Section 1.1 briefly introduces the international standard regulating archive content description, and section 1.2 summarizes the DA concept. Section 2 introduces the ***SIAP*** system containing a **Parliamentary Digital Archive Management System**. In Section 3, we describe the **Web Integrated Architecture** proposed **for the DA**. We remark here on the ontological aspects that we have considered, applied to some archival standards (DCMI, ISAD, ISAAR) and SIAP system. Finally, the conclusions are given.

[2] SIAP *Parliamentary Information Management System*, a CRC Information Technologies product (www.crcit.es)

1.1. Archives

Archives constitute a worthy and representative application area. They contain a huge amount of documentary information on human activity. Archives hold different contents such as culture, business, living and relaxing. However, their main and common goal is: *the propitiation of an easy access to the information content, guaranteeing the safety and custody of what they hold*. Consequently, when you want people to access archive content (such as a catalyst of cultural and historical heritage), undoubtedly the web is the optimal media for a universal safe propagation of content.

The General International Standard Archival Description ISAD(G) (2nd ed.) [24] is the most important one for archivists. Its main goal is the identification and description of the content and context of archive material, in order to promote its management and accessibility. ISAD(G) is developed by the International Council on Archives (ICA) [24], a worldwide organization of archival community professionals. ISAD(G) defines a set of elements (metadata terms) for archive description that may be applied regardless of the format of the archive material. Some combinations of these elements constitute the unit of description.

Figure 1 represents the ISAD hierarchical unit of description levels. Every level has different degrees of detail. Additionally, the International Standard Archival Authority Record for Corporate Bodies, Persons and Families ISAAR(CPF) is a second one. Section 3 describes how to use these two standard concepts. (http://www.ica.org for ISAD and ISAAR)

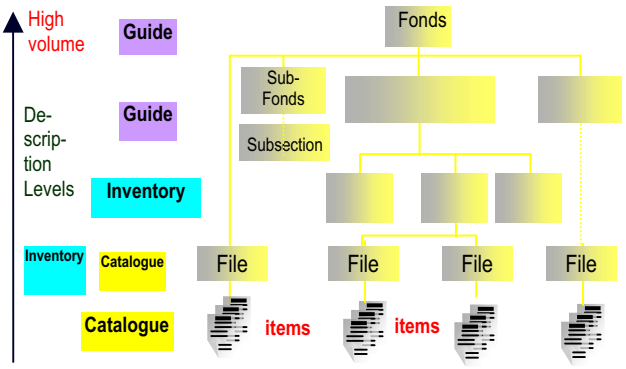


Figure 1. ISAD(G) Hierarchical Description Levels of Archive Organization.

1.2. Digital Archives

The **DA** holds a collection of **documentary data** in digital format joined to **descriptive data** on the organization, its goal and the content of these documents. The descriptive data are **metadata** for archive description at any level that ISAD(G) proposes, as shown in figure 1.

Any DA can store many different contents (corporate databases, data warehouse, thesaurus for information retrieval, web pages, e-mails, documents, maps, photos, etc.). It is true that, in some archives, the process of digitizing could be difficult, slow and may be impossible obtain completely. However, currently more and more institutions are trying to do it in order to offer a higher degree of accessibility. Hence, the web DA is disseminating certain contents that, generally, consist of **metadata** such as: descriptive data for indexing thesaurus of *Information Retrieval*,

summaries, index, keywords, synonymous and antonyms lists (*broader terms*, *narrow terms*, etc.) and other related terms [1]. All this information is stored in what we call DAs, as digital data source repositories. Nowadays, this information is what -through web sites- users are accessing in order to get certain relevant aspects on any fixed unit of description combination (item or collection of items, file or collection of files, Sub-Series, Series, Sub-Series and/or Series collections, Sub-Fonds, collection of Sub-Fonds and Fonds).

2. *'Asamblea of Madrid'* Parliamentary Archive

The *Asamblea of Madrid* parliamentary archive has totally digitized its documentary fonds, since the beginning of our democracy. This DA management system is powerful and ideally advanced for the web.

The DA management system runs at the *Asamblea de Madrid* as a part of a broader system called SGP (Parliamentary Management System). SGP was funded by the *Asamblea de Madrid* (1997-2000) and was built at our research group together with people from Spanish businesses ***CRC Information Technologies*** *(CRC IT)*. Nowadays, CRC IT commercializes SGP as ***SIAP*** and this product is sponsored by our Technical University of Madrid (UPM), by our Technical High School of Telecommunication Engineering (ETSIT) and by Oracle and Cronos Ibérica entreprises.

SIAP manages and controls the workflow of overall parliamentary documentary information and automatically creates the many types of parliamentary document produced by political activity. For the ***SIAP*** design we used methodologies [13, 26] and CASE tools; in addition, the main design guidelines were: the proper political activity, institutional norms, Political Initiative typology and the nature of political documentation. In order to provide *useful information*, as intelligent as possible, we follow the guidelines in [3, 12, 45, 46] and our experience from previous systems [10, 11] among others.

The ***SIAP*** system structures Political Initiative and Parliamentary jobs (Plenary sessions, Committees, etc.). It has many applications (www.crcit.es/siap). ***SIAP*** workflow sends the document to be considered established by the regulation, according to each type of political initiative. Moreover, it is controlled when a document is sent to be published into the Official Bulletin, the Sessions Journal and on dynamic web pages automatically created for a specific user type. ***SIAP*** associates the document to the respective File (*Expediente* in Spanish), to the corresponding Daily Order, to the pertinent Official Bulletin, to the previous scripted political sessions, etc. Besides, ***SIAP*** also knows where the original document is stored and where the multiple document copies are held.

Section 2.1 summarizes the more general aspects of ***SIAP***, successfully running at the *Asamblea of Madrid* since 1999, and section 2.2 describes its **Archive Management System.**

2.1. SIAP system: Objectives and Architecture

SIAP models Parliament's overall structure and organization. It has demonstrated a high degree of security and effectiveness for those procedural steps in every political initiative. The main goal of ***SIAP*** is to obtain a perfect integration of the parliamentary information with the proper activity of the Parliament.

SIAP controls input documents and produces a huge amount of output documents, some of them with a public nature. It makes the entire Parliament of Madrid Official Bulletin, and the greatest part of Sessions Journal. Besides, ***SIAP*** applies documentary searching to other external Official Bulletins (from foreign institutions). If it were convenient, all this information would be dynamically published on the web.

-Management of information generated by political Institution' activity,
- Control of regulated workflow,
- Design and implementation of Parliament structure and organization and its perfect integration into the parliamentary documentary flow,
- Semantic control, applications and GUIs software.

SIAP has an open architecture to be implemented into the Institution network. It has an underlying object-relational database running into the Database Server and its applications could also work on its own (Law Budget Project, Archive Management, Registers, etc.).

SIAP runs on Oracle in Client/Server (C/S, two ties) and in Internet/Intranet (three or four ties). In C/S, for providing services to civil servants that update the information system (with a high degree of protection and security). In C/S, protocols SQL*Net8 and OCA and, in Web OAS (Oracle Application Server and IAS of Oracle 9i) are used. For both, IntermediaText for information retrieval (before Oracle Context) is used. Figure 2 represents the documentary workflow and ***SIAP*** open architecture; where BOAM means Asamblea of Madrid Official Bulletin and DDSS means Sessions Journal.

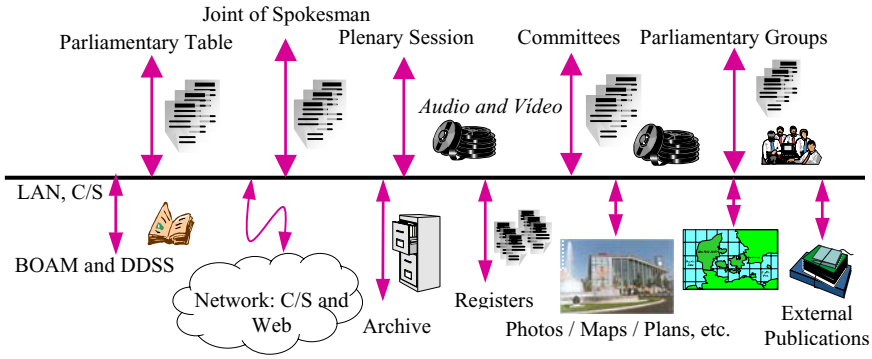


Figure2. Documentary workflow and ***SIAP*** open architecture (C/S and Web)

2.2. SIAP Digital Archive Management System

Parliament activity produces documents tailored to a specific type of Parliamentary Initiative. ***SIAP*** **Archive Management System** holds especially relevant information organized in **files** (descriptive folders) according to the

initiative typology and objective. The **file** collects related information on political activity and marks the procedure type to schedule with all documents that are held inside it (initiatives, photographs, maps, graphics, audio, video, etc.).

Additionally, the file incorporates other descriptive information. Concisely, it includes the following: 1) **Identification**, 2) **Censure or judgment of the contained documents**, 3) **Schedule**, 4) **File Classification and its respective documents**, 5) **Allocation**: topographical, informatics and address of Institution, departments, dependencies, etc., 6) **File prosecution** made up of: a) **Workflow** considering all states that must or should be adopted by the file and any of the related documents annexed to the file, b) **Current status**, c) **History** and, finally, 7) **Relationships** between files by subject, type, date, state, descriptors, etc.

This **DA** handles queries against the underlying object-relational database (**SQL)** and/or to the thesaurus with a powerful **information retrieval**. It searches thousands of publications and locates the subject of interest in a few seconds; locates the annexed document to the file in an almost instantaneous response, independently of the Legislature where it is stored [1]; and, finally, creates a lot of report results [9] (www.asambleamadrid.org).

SIAP controls the all of the information from any Legislature and knows the topographical key signatures of its complete archive fonds. For example, The Madrilenian Court can instantaneously locate all files and documents dealing with *anorexia,* since the beginning of the Spanish democracy and can reproduce video fragments of the political sessions (Plenary and Commissions) in which this illness was discussed.

Through web applications, the parliament can negotiate the information exchange between similar institutions at any level, in an integrated and intelligent way. ***SIAP*** can offer **global information** produced by many others Institutions, as well as its own [20, 36, 40].

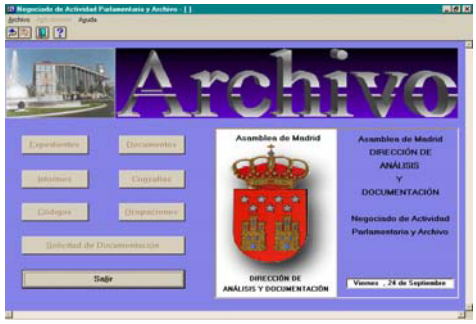


Figure 3. The SIAP Digital Archive at the Asamblea of Madrid

The ***SIAP*** **Digital Archive** was tested for three hours in a ***Parliamentary Archives and Political Parties Conference*** (Nov. 1999) held at the Asamblea of Madrid, and it was considered a powerful and complete system. Figure 3 shows its main screen.

If ***SIAP*** runs in other institutions, web information capabilities will grow in a spectacular way. We want to remark on how easy is to get dynamic interoperability on the web when data sources have the same WIS (same design and semantic control, similar software, etc.) (See Fig. 4). On the other hand, in heterogeneous WIS environments it could be hard to provide easy and powerful access to global information.

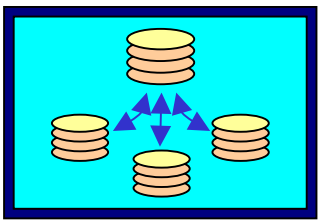


Figure 4. Web Multi-parliamentary

3. Integration of Web Digital Archives

As the current web is inherently heterogeneous, semantic web will be absolutely necessary, mainly where you need a certain conceptual organization. Besides, public DA integration should allow access to inherited cultural content without distances, languages and cultures barriers.

There are many important initiatives based on DA (OAI, DCMI, EAD, etc.) in order to make them accessible to the web user. The systematic and (semi-) automatic generation of web DA allows the administration of the huge documentary legacy held into the Archives to be improved, as well as facilitating the creation of new DA available to any web user.

However, we still do not have a good integrated web solution for making it possible for any DA in the world to be '*added*' to a web environment (integrating several DAs), independently of DA content and description level. A web user is still unable to access a **virtual, global, integrated DA,** whose semantics will be universally guaranteed by a web-integrated system.

Due to the large amount of non-digitized, nor computerized Archives; an *ad hoc* solution to develop a concise DA or to integrate '*n*' certain pre-existing archives is not enough. For this reason, we are seeking a generic solution allowing any DA be published on the web, as well as their generic integration. This kind of integration sends the user queries to 'n' DA just as if they were a single virtual archive.

3.1 Virtual Integration of Web Heterogeneous Data Sources

Research on the dynamic and virtual web integration of multiple heterogeneous DA represents a very different challenge to the classic database static schema integration. The latter is obtained by using a fixed federation of local participant databases schemata in a global static schema [40].

In the virtual integration of web heterogeneous data sources there is neither centralized information management, nor federated information, nor integration information systems dealing with materialized data [15]. In the virtual integrated systems, data sources remain as autonomous data inside every locality. Integration is only virtual (not materialized), as we describe later.

In spite of having many standards in databases and static schema integration reference models, a systematic solution for web dynamic integration, requires a standard that does not yet exist (solutions are often static and *ad hoc*).

However, the question of the non-existence of a standard web is old [20]. Since the first attempts to regularize data exchange on the Internet (EDI) until the proliferation propitiated by the web (http, html, xml, obi, cxml, etc), it has still not been possible to model the interactions among a system components properly [33]. We are dealing with many independent WIS, whose first requirement for becoming part of any integration is to guarantee total autonomy to each local participant in the dynamic and virtual integration (DAs, in our case). Therefore, the DA does not federate in an integrated static global schema. On the contrary, it is produced by a **Web Mediator System** [47]. By doing so, data remains in local data sources, and integration takes place *on-the-fly,* during the query pre-processing in the mediator.

3.2 Introduction to DAWIS-UPM

DAWIS-UPM [6, 43, 18] is a project which global objective is the definition of a web-integrated architecture, providing a virtual and dynamic access to many different DA. DAWIS-UPM is the successor of SIAP, now looking for the web integration of multiple DA. In this sense, we find the acquired experience in the previous project we have carried out for the Parliament of Madrid very valuable.

So, heterogeneous web data source integration and semantic web, applied to DA is the focus of our current research. We are investigating DA conceptual modeling in order to define a virtual web integrated architecture through a unified semantic mediator layer (made up of ontologies, mappings and data repositories) and wrappers (coupling heterogeneity between data sources). Communication and interoperability are based, among others, on web services and XML respectively.

As well as other aspects, DAWIS-UPM needs to provide a systematic and (where possible) automatic definition of the underlying **reference architecture** [6], through which the web user could access a lot of heterogeneous DA [17] transparently. This means the consideration of the following aspects: web architecture, DA model specification, query processing, wrappers, and "mediators".

The generic goal is mainly to provide a semantic unification and web knowledge enrichment. To obtain this, we do not need to start from scratch. There are interesting proposals such as FEDORA and OAI, that offer current interoperable architectures limited to providing remote access and C/S, to a determined DA [19, 38, 39, 42]. Undoubtedly, they are an important first step.

Figure 5 represents the proposed architecture, that will be service-based and compliant to current web-centric architectures (J2EE).

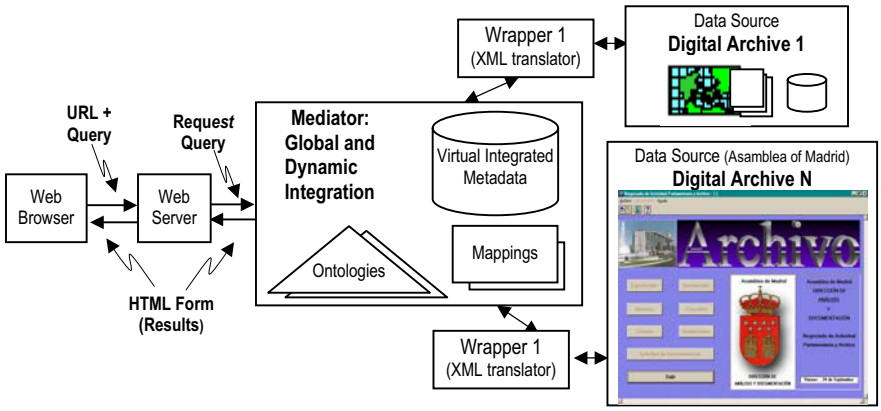


Figure 5. Web Digital Archive Integrated Architecture through Mediator and Wrappers.

We are aware of the achievement of a **Dynamic, Global and Virtual Web DA Integration**. It constitutes an important challenge for current WIS. To give an example, we are researching in order to make it possible for a web user to query something like this: *"Find all documents dealing with equestrian statues of the King Carlos V"*, avoiding the consideration of which physical archive he/she needs to refer to for this query. If it were necessary, the system would be in charge of informing about the origin of each document that makes up the overall answer.

The final idea is the implementation of this architecture as Java libraries based on service-levels [18], where "mediator" and "wrappers" will be dynamically generated starting from the web user query. In order to achieve these goals, it is necessary to consider the following architectural topics: flexibility, integration and their related semantic problem (considered in section 3.3), interoperability and cooperation, and efficiency, as we discussed in [6].

We want to remark that a web services architecture does not solve the semantic integration problem [4, 29]. Web services allow data between remote applications to be exchanged, but they do not guarantee that the receiver application will be able to understand them. As pointed out in [44], the problem is the lack of a global ontology.

Since the Web Application term appeared (WebApp) [33], its capabilities have been growing continuously. For example, initially some WebApp supported only static pages, while now, in J2EE mark, a combination of Servlets and JSPs allows to generate dynamic pages.

In ***DAWIS-UPM***, Servlets are of special interest, because they have a certain semantic orientation. Servlets are Java classes for request processing and corresponding answers generation, supporting part of the application logic (although it will be only in an explicit way) [29]. Moreover, CORBA is another standard that needs a wide range of services providing design and development of current distributed systems [4]. However, the puzzle is not yet complete. Again, the weakest part is the *semantics of the systems*, business logic rules of any participant application. Therefore, it is not surprising that it were in the server applications environment where the alarm first sounded. Thus, in order to solve the dynamic and virtual web integrated access problem, ***semantics of the systems*** is an important and preliminary characteristic, which must be considered.

Thus, we consider ontologies that contribute to reinforcing these semantic necessities. This consideration is different to current distributed architecture approaches, which only consider defined static interactions between software components (during the compilation) and between programs and users [41].

As well as the semantic problem, it is also necessary to provide mediation services [47] for web query processing [32]. Considering figure 6 and coming first from the front-end browser to the back-end DA, the next main services are needed:

a) From the portal web, the global query arrives at the mediator where the query concepts and relationships are matched to a certain ontology (coded in OWL) and the information that repository holds (pre-processing).

b) The mediator, taking into account the related mappings with terminological semantic unification, is able to inference knowledge at this virtual integrated level.

c) The mediator, once the user query is pre-processed, and making use of the mappings between mediator concepts and concepts coming from each DA; knows which local queries (as web query components) need to be invoked –via wrapper- for any DA manager execution. Each locality returns the answer to the mediator, in a well-known format, previously specified by the mediator.

Now, coming from any back-end DA to the front-end browser, ***DAWIS*** should have, at least, services offering the following functionalities:

d) Any DA establishes the query types it wants to offer to the mediator, in order to be a specific dynamic integration participant.

e) In each DA, every XML schema contains the extracted concepts that make its query capabilities possible (tailored to the measurement of each possible local participant integration). These schemata determine the way any DA is taking part of any global architecture.

f) These XML schemas must to be compliant to those "mappings" defined between the "wrappers" and mediator ontology.

Finally, we want to remark here that we are making only a preliminary requirement specification on the necessary web architecture. This justifies our contribution to the semantic web focused on DA application that we discuss in the rest of this paper.

3.3. Ontologies in semantic web

According to [2], the web will evolve unavoidably toward a semantic web. Then, computers will find the meaning of data following the *links* to the definitions of the terms and key rules. With this, designing automatic services will be easier; because the main bottleneck, for getting the semantic web, is the modeling of the information. To solve this problem and to be able to code the necessary information, we have XML and ontologies as better options [21].

An ontology describes shared knowledge on a specific domain in order to establish a communication via between humans and programs [1, 8]. Each ontology provides a precise definition of the terms (concepts) of a specific domain; that is to say, it facilitates a common language for sharing and to re-using knowledge of some phenomenon in the domain of interest [21, 49] between applications and groups.

Consequently, applications can speak to each other and interpret, without ambiguity, the information they exchange. For this reason, they have been accepted as powerful description tools that allow to make explicit the semantic web. Since the early 1990s, a lot of research [2, 5, 7, 22, 23, 49] deals with these semantic problems, where ontologies play a fundamental role.

Moreover, XML is a more mature technology than ontological languages (DAML+OIL, OWL) because of the amount of user communities and available tools (http://sws.mcm.unisg.ch/links.html#daml). Kim affirms that ontologies should be adopted where the capability to represent semantics will be as necessary as forgetting the maturity advantages that XML already offers (http://www.xml.org/) [27].

We need to build ontologies [48] because DA domain is not simple and it needs to be used for a wide audience. We adopt ontologies as a paradigm that reduces complexity, and XML is the technology used for the interoperability of archive data.

3.4. Ontologies in DAWIS

As with the aforementioned architectural aspects, the archival conceptual modeling does not start from scratch either. On the one hand, we have the well-known DCMI (Dublin Core Metadata Initiative) ontology. On the other hand, ISAD(G) and ISAAR(CPF) are international standards specifying how archival

content should be described. These standards are widely followed by archivists. Finally, a fourth special Parliamentary ontology - coming from the SIAP Parliamentary DA - is also being undertaken.

All these items will be considered for achieving the wide ontological unification of DA domain that we are looking for.

DCMI Ontology

The Dublin Core Metadata Initiative (DCMI) [14] promotes metadata standards and develops specialized vocabularies for describing resources. According to the RFC 2396, a resource is "something that has identity".

The DCMI proposal was came about in 1995 inside the Online Computer Library Center (OCLC). Since then, DCMI collaborates with the normalization and the development of technologies that allow a bigger efficiency of the use of metadata. Today, DCMI is a pillar inside a semantic web, an obligatory reference to all ontology groups that want to adopt a common standard for the semantic description of any very general resource. DCMI ontology was formally adopted by the European Committee of Normalization. In 2001 it was accepted as the ANSI/ISO Z39.85 norm and, since February of 2003, it is the ISO 15836-2003 standard.

Although DCMI is not a specific ontology for describing archival material, it could be perfectly adapted to describe any kind of resources. Since 2000, DCMI is already a free ontology that we have taken through the Protégé tool.

ISAD(G) & ISAAR Ontologies

Since 1999, ISAD(G) has been widely used as unquestionable classification base of national archives [31]. Standardization models play a very important role for the future success of accessibility. ISAD(G) defines twenty-six descriptive elements (metadata terms) specific to archival description. Only six of these elements are considered essential and must be used in any archival unit of description.

ISAD(G) specifies the identification, content and context of archival information units. It also provides links in order to fix any combination of archival information elements. One of the main archivist activities is the description of record creators, a difficult job in documentation and maintenance. In this sense, it is good practice to

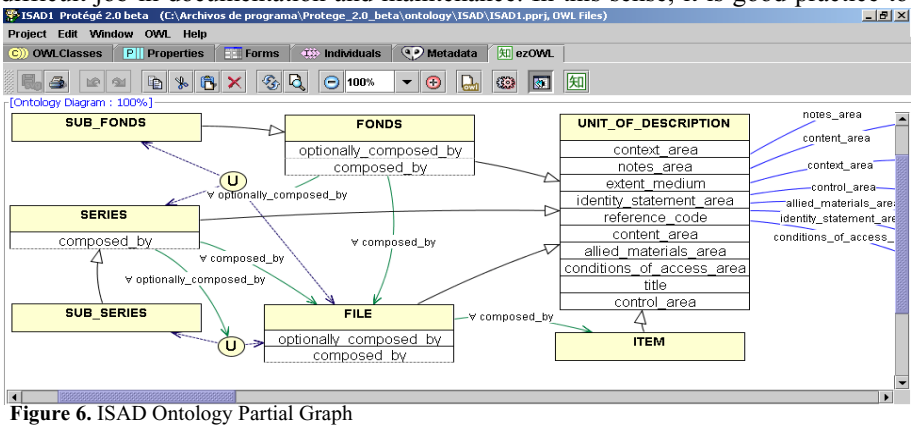


Figure 6. ISAD Ontology Partial Graph

independently record the creator(s) of a unit of description and then link this to the related unit of description. This practice enables the creator description to be shared

between different archival documents, or repositories, only adding linking information.

Moreover, ISAAR(CPF) [25] can be considered as a complement to ISAD(G) to specifically manage the description of authority records. It defines twenty-seven descriptive elements; where the concept "Authority record" is similar to the concept of Unit of Description in ISAD(G). In this sense, an Authority Record describes an archival authority (Corporate Bodies, Persons and Families) as well as the Unit of Description describing an archival unit of information.

Regarding ISAAR(CPF) and ISAD(G) elements, in [43] we have defined the UML respective models. Starting from these, we have developed ISAD(G) and ISAAR(CPF) ontologies. Figure 6 shows a partial graph of this ISAD(G) Ontology.

SIAP Ontology

Both, ISAD(G) and ISAAR(CPF), propose general rules that can be adapted to describing any archive; nevertheless, these rules are not considered for the peculiar description of special archives. In our case, Parliamentary archives have a part of their concept that are particular, due especially to the workflow that has to follow each Parliamentary file and peculiar parliamentary organization. In this sense, we have developed a SIAP ontology that comprises particular Parliamentary concepts.

In order to define this specific ontology, we start from the ***SIAP*** DA conceptual model (see Figure 7). Parliamentary DA type has certain particular information, and also specific description rules, where the semantics of the terms and the elements can be different from other kinds of archives.

But the interest extends further on, because -in the future- the integration way, here considered, will allow us to conceive a system able to offer rules to integrate these specialized concepts of a diverse nature (police, government, etc.). Ontologies are complex to build all in one go. So, building them bit a bit, testing each new (kind of) addition empirically and developing appropriate learning techniques for each bit, you may automate the process; and, next time, you may build a new one in a more systematic way.

Summarizing, we have the DCMI ontology (free on the web) and three original

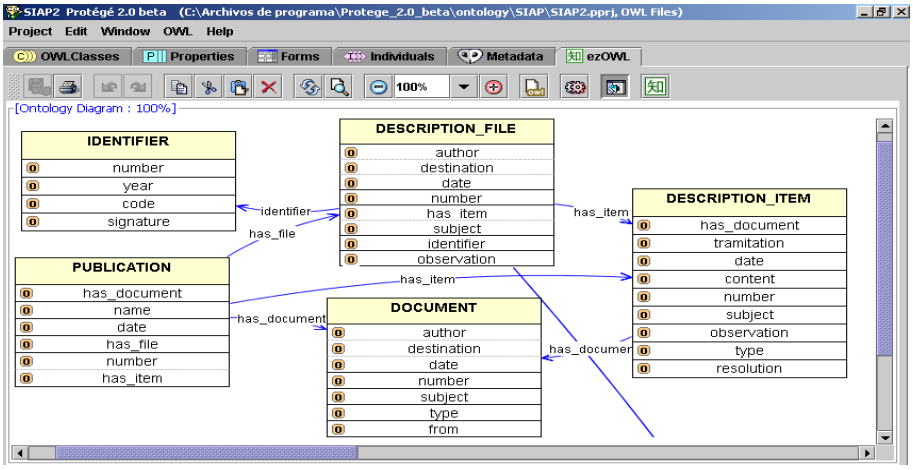


Figure 7. SIAP Ontology Partial Graph

ontologies based on ISAD(G), ISAAR(CPF) and SIAP that we have developed [21, 28]. The majority of current semantic web is built in this way. In fact, ontologies are now singular web "*hot-spots*" whose semantics are bigger than usual. Today, that web has several hundred of these isolated ontologies [49]. We consider this kind of semantics as a worthy first step that we call the ***single-ontologycal-centric*** approach.

Generally speaking, an important requirement of all different ontologies -closely related to a specific domain- should converge in only one semantic unification and, where possible, with a universal scope. In our DA case, it will allow a semantic, powerful and really open integrated web ontology inside the mediator layer to be guaranteed [22].

3.5. Ontological Integration Kernel

According to the previous semantic unification we have stated, this section discusses our preliminary approach concerning to the definition and development of a global ontology [49], unifying the semantics of the diverse existent ontologies related to DA domain [5].

Regarding this, we consider two important previous works: *Observer* [34, 35, 37] (http://sol1.cps.unizar.es:5080/OBSERVER) and *Harmony* [16, 30] (http://metadata.net/harmony). They constitute an important contribution to our current research.

Observer considers ontologies as if they were on only one horizontal plane. Then, inter-ontological mappings could be also considered in a parallel plane. For this reason, we call this kind of relationships "***inter-ontological-horizontal-mappings***". As well as this, *Observer* provides the capability to query many specific ontologies from different domains. Only if the user wants, the system allows a new query formulation referring other ontology concepts. In this sense, the answer enrichment is provided. We want to recognize the *Observer's* great improvement regarding query processing against many ontologies.

On the other hand, *Harmony* -a digital library project- defines a common "*ABC metamodel*" for metadata integration coming from many other ontologies. *Harmony* considers the *ABC model* as if it were on a higher plane, acting as an umbrella (containing very general concepts), allows other more specific ontologies to be attached on a lower plane (CIDOC/CRM, MPEG7, IMS,). Then, inter-ontological mappings are always provided on a vertical plane. For this reason, we call this kind of relationship "***inter-ontological-vertical-mappings***". Again, we want to appreciate this valuable contribution.

Considering these two project results as pillars, we propose to develop specialized techniques and methodologies that can guarantee a correct **shared and common semantic integrity between current DA ontologies**. Our ontological integration kernel implementation will be carried out on the definition of an ontological unification approach, operating as a kernel in which current (more or less) specialized ontologies could be semantically connected, as figure 8 represents.

The distinctive feature ***DAWIS*** presents is the semi-automatic ontological kernel generation for a global integration that does not have to be defined in either a manual or static way. Thus, this ontological kernel will be in charge of providing a common understanding for fundamental DA concepts. In order to obtain a flexible

and scalable integration, "bridges or steps" between ontologies - such as an intelligent and generic "crosswalk"- need to be provided. This allows knowledge inference from any ontology to another.

We are aware that inter-ontological semi-automatic mapping definition can turn out to be a difficult and expensive task, caused by the semantic heterogeneity. So, today, our research is continuing in this direction applying the bit-by-bit empirical methodology we have previously mentioned.

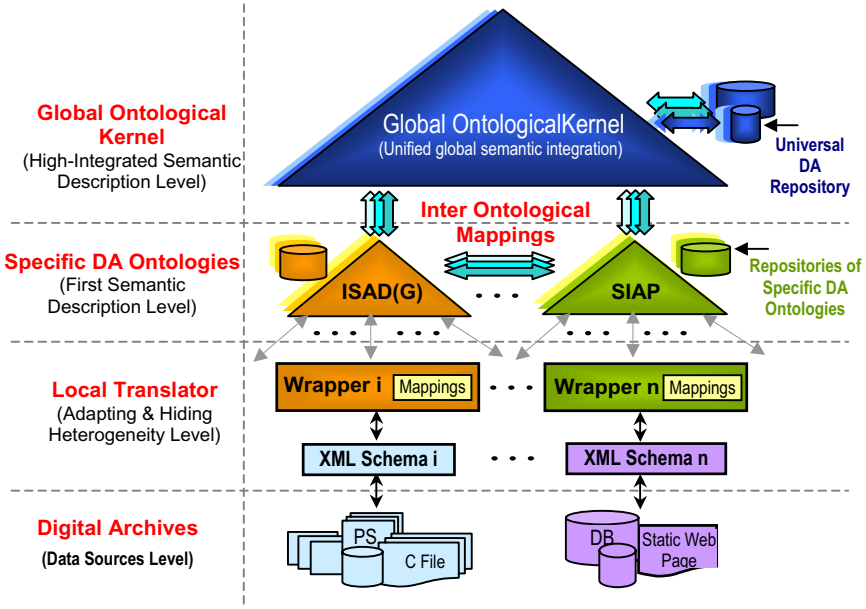


Figure 8. Integration model based on the ontological unification of the mediator

4. Conclusions

This paper focuses on current Web Information Systems (WIS) applied to Digital Archives (DA) looking for a virtual, flexible and dynamic integration by means of a Mediator and Wrapper architecture. We start from the Parliamentary Integrated Management System that has been running at the *Asamblea of Madrid* since 1999. After this, the paper presents four current ontologies closely related to DA domain. Applying an empirical step-by-step methodology, a high-integrated semantic description level is proposed by means of a global ontological kernel inside the Mediator layer.

This research is still in its early stages. However, we consider the adopted empirical method as a preliminary corner stone, to establish a solid foundation for the achievement that this web architecture is looking for (Java libraries and web services).

Acknowledgments

The authors would like to thank the anonymous reviewers. Their valuable comments and suggestions greatly improved the quality of this paper.

This work is partially granted by the Ministry of Science and Technology (MCYT-TIC2002-04050-C02-02, DAWIS-UPM project), by Community of Madrid (07T/0056/2003/3, EDAD-UPM project) and by Spanish NoE Databases, RedBD, MCYT-TIC2000-3250-E.

References

[1] Baeza y Berthier, *Modern Information Retrieval*. Addison Wesley, 1999.
[2] Berners-Lee T, Hendler J, Lassila O, *The semantic Web*. Scientific American 284, 5, pp.34-43, May 2001.
[3] P. Bernstein, A. Halevy and R. Pottinger, *A Vision of Management of Complex Models.* ACM Sigmod Record, pp. 55-63, Vol. 29, N. 4, Dec. 2000.
[4] Brittenham P, *Web Services Development Concepts (WSDC 1.0)*, IBM Software Group, May 2001
[5] Calvenese D, Giacomo de G and Lenzerini M, *A Framework for Ontology Integration*. In Proc.of the First Semantic Web Working Symposium, pp 303-316 – 2001.
[6] Costilla C, Eibe S, Menasalvas E, Sáenz J, Marcos E, Cavero J y Vela B, *DAWIS: Enfoques Preliminares sobre la Arquitectura de Referencia para la Integración de Archivos Digitales en Web*, Taller de la Red de Excelencia de Bases de Datos en España (RedBD) dentro de VII JISBD'02, El Escorial, Madrid, Nov. 2002.
[7] Chandrasekaran B, Josepheson J, Benjamins VR, *Ontologies: What are they? Why do we need them?* IEEE Intelligent Systems, 14(1):20-26, 1999.
[8] Clyde W, Hossapple, Joshi KD, *A collaborative approach to Ontology design*. Communications of the ACM. Vol. 45 n. 2, pp.42-47, Feb. 2002.
[9] Costilla C, Calleja A, Cremades J, *SIAP: Sistema de Información para Ayuntamientos y Parlamentos*, Revista Círculo de Usuarios de Oracle, CUORE, Sección *'Vivat Academia'*, 8 pages, Oct. 2003.
[10] Costilla C, Bas MJ, Villamor J, *SIRIO: A Distributed Information System over a Heterogeneous Computer Network*. ACM SIGMOD RECORD, Vol. 22, N. 1, pp. 28-34, March 1993.
[11] Costilla C, Bas MJ, Villamor J, *The Object Data Model and its Graphical User Interface for an Object Database System*. Proc. of Data Management System' Biwit'95, pp. 161-172, IEEE Computer Science Press, 1995.
[12] Costilla C, *A Contribution to Knowledge Communication in Distributed Knowledge-Based Systems*, in *Research into Networks and Distributed Applications* (R.Speth ed.), North-Holland, pp.1153-1162, 1988.
[13] Costilla C, *Sistemas de Bases de Datos. Conceptos, Técnicas y Lenguajes*, ed. Serv. Publicaciones ETSI Telecomunicación-UPM, ISBN: 84-7402-271-1, Madrid, 1999.
[14] DCMI. *Dublin Core Metadata Element Set, V. 1.1: Ref. Desc.* Feb. 2003 http://dublincore.org/documents/2003/02/04/ dces/
[15] Domenig R, Dittrich KR, *An Overview and Classification of Mediated Query Systems*. ACM SIGMOD RECORD, Vol. 28, N. 3, pp. 63-72, Sep.1999.
[16] Doerr M, Hunter J and Lagoze C, *Towards a Core Ontology for Information Integration*, JODI 4(1), April 03
[17] Doan A and McCann R, *Building Data Integration Systems: A Mass Collaboration Approach,* 18th Int. Joint Conf. on Artificial Intelligence (IJCAI), 2003. Aug. 2003, http://www.isi.edu/info-agents/workshops/ijcai03/papers
[18] Eibe S, Costilla C, Menasalvas E y Acuña C, *DAWIS: Una Arquitectura de Integración Web para el Acceso Integrado a Archivos Digitales*. VIII Jornadas de Ingeniería del Software y Bases de Datos, JISBD'03, pp. 583-591, Alicante, Nov. 2003.
[19] *The Mellon Fedora Project: Digital Library Architecture Meets XML and Web Services,* Forthcoming European Conf. on Research and Advanced Technology for Digital Libraries, Rome, Sept. 2002.
[20] Florescu D, Levy A and Medelzon A, *Database Techniques for the World Wide Web: A Survey*. ACM Sigmod Record, Sept. 1998.
[21] Gruber, T. R. *Towards principles for the design of ontologies used for knowledge sharing*. Int. Workshop on Formal Ontology, 1993.

[22] Guarino N and Welty Ch., *Evaluating Ontological Decision with ONTOCLEAN*. Communications of the ACM. Vol. 45 n. 2, pp. 42-47, Feb. 2002.
[23] J. Hammer, J. McHugh, H. Garcia-Molina, *Semistructured Data: The TSIMMIS Experience*, Proc. I East-European WS on *Advances in Database and Information Systems*, ADBIS'97, St.Petersburg, Russia, Sept. 1997
[24] International Council on Archives, *ISAD(G) General International Standardization Archival Description*, 2nd ed, ISBN 0-9696035-5-X, Ottawa 2000. http://www.ica.org/biblio/cds/isad_g_2e.pdf
[25] *International Standard Archival Authority Record for Corporate Bodies, Persons and Families,* 2nded. Feb. 2003 http://www.hmc.gov.uk/icacds/eng/ISAAR(CPF)2. pdf
[26] Jacobson, Booch y Rumbaugh, *The Unified Software Development Process*. Adison Wesley, 1999.
[27] Kim, H. *Predicting how ontologies for the semantic web will evolve*. Communications of the ACM. Vol. 45, n. 2, pp. 47-51, Feb. 2002.
[28] Knoblock C, Minton S, Ambite JL, Ashish N, Modi P, Muslea I, Philpot A, and Tejada S, *Modeling web sources for information integration,* Proc. 15th National Conf. on AI and Tenth Innovative Applications of AI Conf. (AAAI-98), pp. 211-218, Wisconsin USA 1998
[29] Kreger H, *Web Services: Conceptual Architecture (WSCA 1.0)*, IBM Software Group, May 2001.
[30] Lagoze C and Hunter J, *The ABC ontology and model*, Journal of Digital Information, JODI, Nov. 2001, 2(2), http://jodi.ecs.soton.ac.uk/Articles/v02/i02/Lagoze/
[31] Landis B, *What is ISAD(G)?,* Description Section, Society of American Archivists, 2000.
[32] LeFurgy, *Levels of Service for Digital Repositories*, D-Lib Magazine, 8(5), May 2002.
[33] Leymann F, *Web Services Flow Language (WSFL 1.0)*, IBM Software Group, May 2001.
[34] Mena E, and Illarramendi A, *Ontology-Based Query Processing for Global Information Systems*. Kluwer Academic Publishers, July 2001.
[35] Mena E, *Observer: An Approach for Query Processing in Global Information Systems based on Interoperation across Pre-existing Ontologies*. Tesis Doctoral, Universidad de Zaragoza, 1998.
[36] Mendelzon A, Mihaila G and Milo T, *Querying the World Wide Web*. Journal of Digital Libraries, (1,1) 1997.
[37] Mena E, Illarramendi A, Kashyap V, and Sheth A, *OBSERVER: An Approach for Query Processing In Global Information Systems Based on Interoperation across Pre-Existing Ontologies*, Distributed and Parallel Databases, vol. 8, pp. 223-271, 2000
[38] Open Archives Initiative. *Implementation Guidelines for the Open Archives Initiative Protocol for Metadata Harvesting Protocol* Version 2.0 of 2002-06-14 Document Vers 2002/06/13T19:43:00Z
[39] OAI, *The OAI Protocol for Metadata Harvesting Protocol* Vers. 2.0., Feb. 2003 http://www.openarchives. org /OAI/2.0/ openarchivesprotocol.htm
[40] Özsu MT, Valduriez P, *Principles of Distributed Database Systems*, 2nd ed, Prentice-Hall, 1999.
[41] Payette S and Lagoze C, *Flexible and Extensible Digital Object and Repository Architecture,* Second European Conference on Research and Advanced Technology for Digital Libraries, Heraklion, Crete, Greece, Sept. 21-23, LNCS, Vol. 1513, Springer, 1998
[42] Prom CJ and Habing TG, *Using the Open Archives Initiative Protocols with EAD*, JDCL'02, July 13-17, 2002, Portland, Oregon, USA, 2002.
[43] Sáenz J, Costilla C, Marcos E y Cavero J, *Una Representación en UML del Metamodelo Estándar ISAD(G) e ISAAR(CPF) para la Descripción de Archivos Digitales*. VIII Jornadas de Ingeniería del Software y Bases de Datos, JISBD'03, pp. 519-528, Alicante, Nov. 2003. http://sinbad.dit.upm.es
[44] Shirky C, *Web Services and Context Horizons*. IEEE Computer, pp. 98-100, September 2002.
[45] Wand Y, Storey VC and Weber R, *An Ontological Analysis of the Relationship Construct in Conceptual Modelling*. ACM Transaction on Database Systems, pp. 494-528, Vol. 24, N. 4, December 1999.
[46] Wiederhold G, *Weaving Data into Information*. *Database Programming & Design*, Freeman pubs, September 1998.
[47] Wiederhold G and Genesereth M, *The conceptual basis for mediation services*. IEEE Expert, (12,5), pp.38–47, Sep.-Oct. 1997
[48] Wache H, Vögele T, Visser U, et all., *Ontology-Based Integration of Information –A survey of Existing Approaches*. Proc. IJCAI-01 Workshop: Ontologies and Information Sharing. Seattle, WA, 2001, pp 108-117. http://www.isi.edu/info-agents/ workshops/ijcai01/ papers
[49] Zhong N, Liu J, Yao Y (eds.), *Web Intelligence*, Springer Verlag, 2003

SCIENTIFIC PAPERS UNIVERSITY OF LATVIA. 2004. VOLUME 672, Datbases and Information Systems
497–509 P.

Supporting Security in an Electronic Market System on the Base of Web Services

Michael Christoffel, Moritz Killat

Institute for Program Structures and Data Organization
University of Karlsruhe
76128 Karlsruhe, Germany
{christof, killat}@ipd.uka.de

Abstract. Security is a precondition for the success of an electronic market system. Unless the system is able to provide some mechanism to ensure authentication, authorization, and secure data transmission, the system will have problems to be accepted by both customers and providers. In this paper, we will discuss some ideas how to add security to a distributed electronic market system that is designed platform-independently on the base of web services. We will show how we have implemented these ideas in a system for an electronic market for scientific literature.

Keywords. Security, Electronic Market, Web Services, Digital Libraries

1. Introduction

In recent years, the mention of problems in the field of computer security has become quite often in newspapers and even in the TV news. They report on dangerous viruses, computer crime, new security holes in standard software products, and network attacks that make a company's computer system collapse. Practically every Internet user feels the consequences of security lacks day after day: The masses of spam email sent every day attest a misuse of personal data.

Every computer system should be able to consider security systems in some way. This is especially true for electronic commerce systems. Hereby, customers and providers have the same intentions. Customers do not want other people monitor what they are doing, and they want their personal and financial data kept secret. Companies want to provide reliable services in order to content their customers, and, of course, they do not want that others know their company secrets.

However, the big handicap is that the Internet is not secure at all. It is relatively easy to intercept a message while it is transported through the Internet. The message can be read or manipulated without giving sender or receiver the chance of notice. It is even possible to fake complete messages.

Network security encompasses four properties:

- Authentication: Both sender and receiver of a message must be able to clearly identify themselves.
- Authorization: Each person allowed to access the system can do this only within the range of the given rights or privileges.

- Privacy: Private data must not be given in the hands of any other person without the permission of the owner.
- Integrity: Messages transmitted between sender and receiver must not be changed during transmission.

It is not possible to guarantee the latter two properties in an Internet application. However, it is possible to weaken the properties without losing the practical consequences:

- Privacy: In the case that a person's private data falls in the hands of a third person without the permission of the owner, the third person must not be able to use these data.
- Integrity: When a message has been changed during transmission, the receiver must be able to notice the change. Therefore, he/she will be able to throw the message away and ask the sender for a new transmission.

In this paper, we will show how to implement these security properties in an electronic market system, namely the UniCats system which aims to assist a market of scientific literature [3].

The work presented in this paper is supported by German Research Association (DFG) as a part of the national German research initiative "Distributed Processing and Delivery of Digital Documents (V^3D^2)".

We will continue as follows: In the next section, we will shortly describe the UniCats system in the stage before we added security concepts. Then we will discuss the cryptographic algorithms and standards which can be used for the solution of the security problems. In section 4, we will introduce the main ideas of our security improvements, and in section 5, we will describe how we have realized these ideas. In section 6, we will shortly mention some related approaches. We will conclude this paper with a summary of the main points of this work and a discussion of future work.

2. The UniCats System

Background of the UniCats[1] approach is the investigation of information markets. Information has an important role in the modern society, where many professions depend on a steady supply with actual information.

The application domain are markets of scientific literature. These markets have received much attention in the last years due to the success of the Internet, which allows publishing houses, booksellers, and libraries to offer their goods and services world-wide. Additionally, we can observe the growth of completely new kinds of information providers such as delivery services, bibliographic databases, and citation servers. At the same time, the amount of scientific literature demanded by students and scientists at university and research institutions has increased rapidly.

However, the customers in these markets are confronted with the problem of information overload. Often they are not able to survey the multitude of information

[1] a Universal Integration of Catalogues based on an Agent-supported Trading and Wrapping System

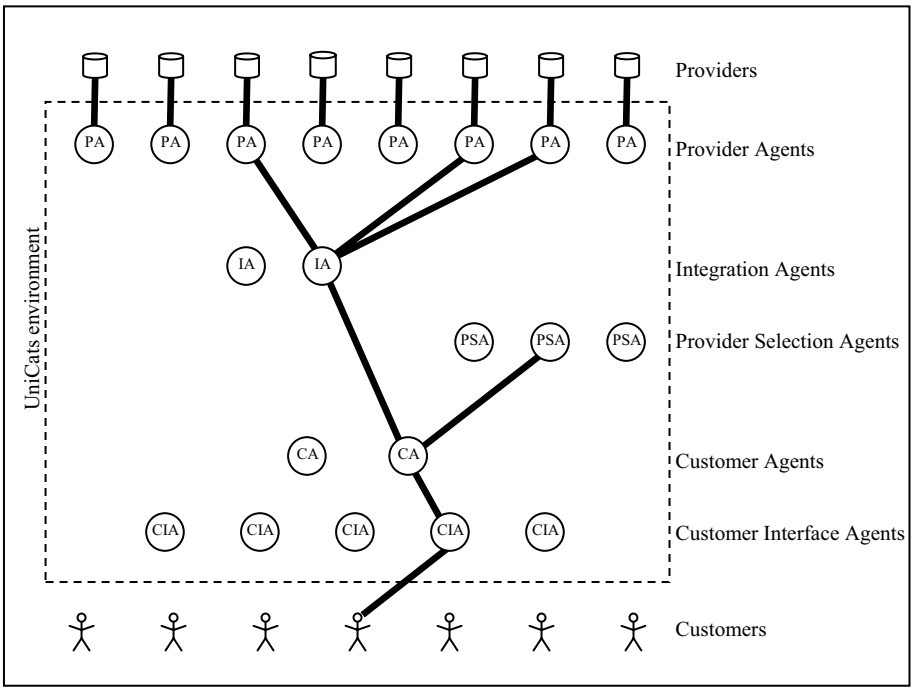


Figure 1. The UniCats environment

sources available or to find the providers most appropriate for their demands. In order to proceed with their search task, they have to apply several services in a sequence, evaluate and compose search results, and find ways of document delivery. Doing all this manually can be expensive in time and money.

The UniCats approach intends to develop a system for the integration of information services and sources, handling heterogeneity and distribution, automating search processes, and providing the customers in these markets a uniform access to the information available [4].

We have developed an agent-oriented platform for the support of an open information market, which we refer as the UniCats environment. In our market model, both customers and providers are autonomous and independent and can enter and leave the market on their own decision without further notice. A discussion of this market-oriented approach can be found in [1]. Although we have considered only markets of scientific literature until now, both the market model and the agent platform are general enough so that they could also be applied to other scenarios and application domains.

Figure 1 contains a simplified view on the UniCats environment, showing the most important agent types. Customer Interface Agents (CIA) are the connection of the customers to the system. This connection is established by a set of user interfaces that can be customized to the customers' personal preferences. Different user interfaces can be used in different situations. So a customer might have a favorite user interface that uses all the capabilities of his/her desktop computer, but

will use a different user interface when he/she will have to contact the UniCats system through a mobile devise during a business travel.

Customer Interface Agents are connected to customer agents (CA) that act as the representatives of the customers in the system and provide a personal workplace for each customer. In order to perform queries in behalf of the customers, Customer Agents can ask a Provider Selection Agent (PSA) for recommendations on those providers, which are most appropriate for the customer's demand.

A Customer Agent sends a query together with a list of the selected providers to an Integration Agent (IA). The Integration Agent sends the query in parallel to the Provider Agents (PA) that act as the representatives of the providers. The Provider Agents translate the incoming query into the native protocol of the provider and re-translates the delivered results into the common protocol. The Integration Agent collects the incoming results from the different information sources and integrates them to a uniform result list. The final result list is sent back to the Customer Agent, which can use the results for the further task execution or present the results to the customer with the help of the Customer Interface Agent.

It is important to consider that this scenario is only an example of possible interactions. A more complex scenario may contain several customers who operate with the system at the same time, require the combination of different queries including order and delivery, and involve many different agents.

In addition to the agents shown in Figure 1, we have implemented some more agents: Customer Organization Agents (COA) represent the interests of customer organizations (such as universities, research institutes, and companies) and their members. Billing Agents (BA) manage financial transactions among the agents of the environment. External Payment Agents (EPA) provide interfaces to financial institution for the access to external accounts and methods of digital payments. Agent Naming Agents (ANA) and Group Naming Agents (GNA) provide a naming service for agents and groups of agents. System Administration Agents (SAA) monitor the entire environment (or a part of the environment) and assist the system administrator in case of a failure.

It is possible to add new agent types that can be derived from existing agent types or from the agent basic class.

For a more detailed description of the UniCats environment see [3].

We implemented the UniCats environment using Java programming language. This brings the advantage that our agents can run platform-independent on most computers. There is also a large set of free tools and class libraries available. However, the development of agents does not depend on the chosen programming language. It is also possible to implement agents using other platforms and languages, and these agents will work together in one environment. We tested this with a sample environment encompassing different hardware platforms and agents written in seven different programming languages.

The UniCats environment is a distributed system that can encompass several computer nodes. Each computer node can hold one or more UniCats communities.

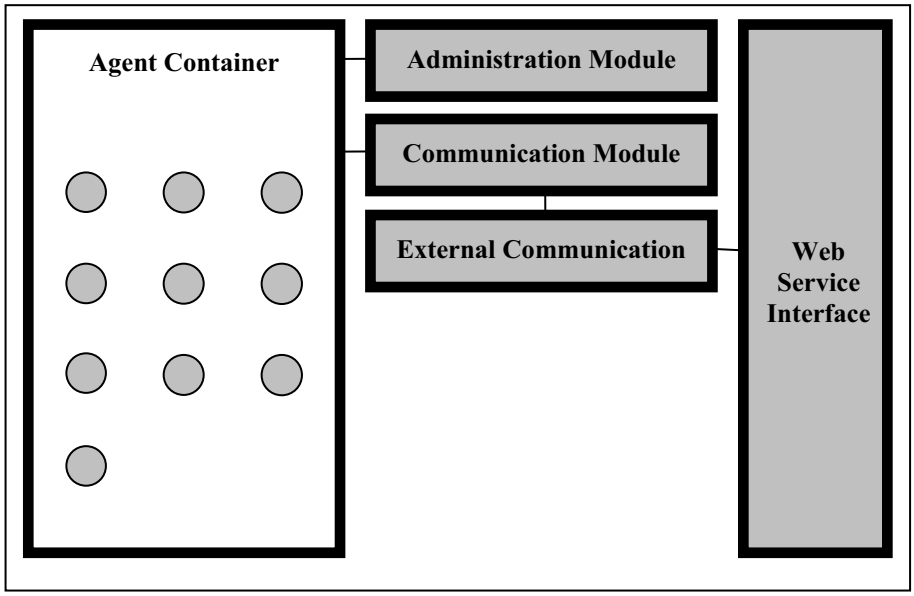


Figure 2. The UniCats community (without security extensions)

Figure 2 shows the basic structure of a UniCats community. The community consists of an agent container and three modules: Administration Module, Communication Module and External Communication Module. The agent container can hold any number of UniCats agents, sharing resources. Agents can be added and deleted at runtime. It is also possible for agents to migrate from one agent community to another.

The administration module is the main module of the community. It is responsible for the initialization of the community and controls startup and shutdown of the agents. The communication module is responsible for the communication of all agents in the community and manages outgoing and incoming messages. There are four different ways of message interchange among the agents:

- Agent communication is used between two agents.
- Group communications is used between an agent and a group of agents.
- Community communication is used between an agent and all the agents in a community.
- System communication is used between an agents and a community itself.

While messages directed to an agent inside the own community are forwarded to this agent on a direct way, the Communication Module delegates messages that are supposed to be delivered outside the community to the External Communication Module. Similarly, the External Communication Module receives all messages coming from outside the community and forwards them to the Communication Module.

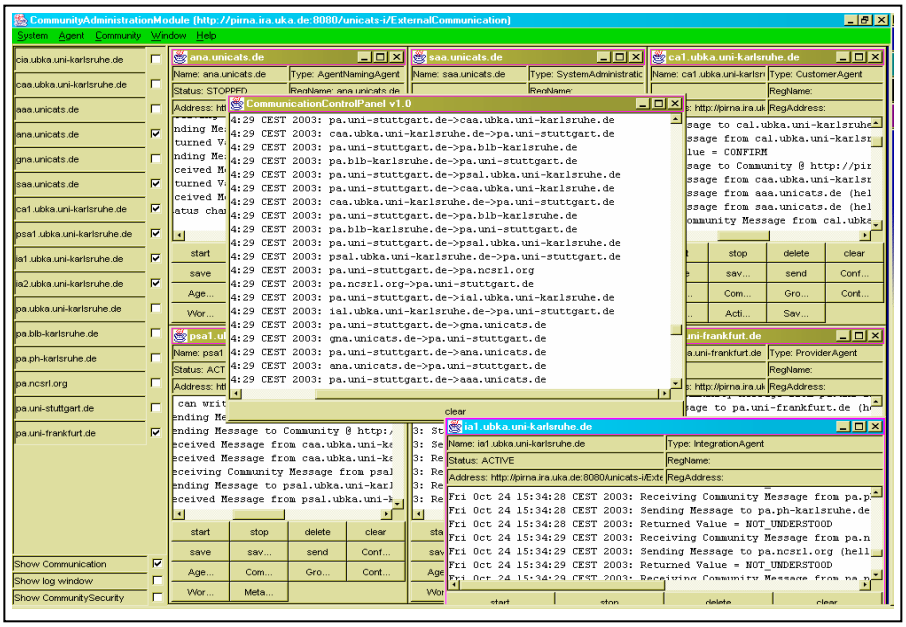


Figure 3. Administration control panel of a UniCats environment

Each module and each agent has a graphical interface, the control panel. The control panels are used to survey and administer the agents of one community. They are hierarchically structured with the administration control panel as the parent frame (Figure 3). In addition to the direct control through the control panel, it is also possible to configure the agents and the community with human-readable configuration files. Important commands can also be applied through a text-based command prompt.

Any communication between different UniCats communities is operated by web services. Each community has a Web Service Interface which is controlled by the External Communication Module. This way, every transmitted message – including all parameters – is automatically converted into an XML document which is delivered to the receiver web service using standard Internet protocols. The use of web services as transport layer is the main reason for the ability of the UniCats system to build cross-platform applications. Another advantage is that we can overcome the firewall problem. While many network administrators close Internet ports for safety reasons, UniCats is not touched by this, because the web service communication uses only those standard ports that are accessible at every system.

However, we have seen that the Internet does not provide privacy. Messages transmitting as XML documents through the Internet are free for unauthorized reading and manipulation. Moreover, the openness of the UniCats environment allows agents and market participants to penetrate uncontrolled into the system. If UniCats should really be recognized as a platform for electronic commerce, we need

a solution for these security problems. Before we will present the solution we have developed, we will first introduce some basics in cryptography.

3. Cryptography

When a message, which content is supposed to be kept secret, has to be transmitted through an unsafe medium (such as the Internet), then it is not possible to exclude the danger that this message comes in the hand of a third person. A solution for this problem is to use encryption: The sender transforms the message into a cipher text, and only a receiver who knows the right key can re-transform the message into a readable form. For all other persons the cipher text is useless.

The easiest way to establish encryption is symmetric encryption, which uses only one key for both encryption and decryption. Principally, symmetric encryption bases on a revertible XOR operation between message and key. One of the most popular symmetric encryption algorithms is DES (Date Encryption Standard), which has been standardized in the year 1977 and uses a 56 bit key. However, due to the relatively short key length and the increased power of the contemporary computer systems, DES can be defeated by a brute-force attack. Therefore, a modified algorithm has been published under the name 3DES (Triple DES), which uses a key length of 112 or even 168 bit. A possible successor is AES (Advanced Encryption Standard), which has been standardized in the year 2000 and can support key length up to 256 bit.

However, symmetric encryption has two major disadvantages. First, it is necessary to have a separate key for each pair of sender and receiver. Second, since both sender and receiver must know the symmetric key, this key has to be transmitted over the network at least one time, giving an attacker the chance of interception.

Both disadvantages can be solved using asymmetric encryption. Here two keys are involved. A message encrypted with the first key can be decrypted with the second key and vice versa. Usually one key is kept secret (private key), while the second key is published (public key). When a sender intends to send a message to a receiver, the sender uses the public key of the receiver for encryption. This way, only the receiver who holds the corresponding private key can decrypt the message. The most popular asymmetric key algorithm is RSA (Riverst Shamir Adleman) from the year 1978, which bases on prime factors. Even today, RSA with 2048 bit key length is supposed to be very safe.

The drawback of asymmetric encryption is performance. Symmetric key algorithms are faster than asymmetric key algorithms, to be exact by the factor 100-1000. Therefore, practical applications use a combined procedure, using asymmetric encryption for the exchange of a symmetric key, and then use symmetric encryption for the message itself.

An alternative way for key exchange provides the Diffie Hellman algorithm from the year 1976, which bases on the problem of discrete logarithms. The Diffie Hellman algorithm enables two communication partners to arrange for a common symmetric key by an iterative protocol. With the Diffie Hellman algorithm it is not necessary to transmit the symmetric key at all, and this algorithm is much faster than

asymmetric key algorithms. However, the Diffie Hellman has another disadvantage: it does not stand a man-in-the-middle attack. For an attacker, it is possible to interrupt the communication between the two original communication partners and make both sender and receiver arrange common symmetric keys with him/her.

Another application for asymmetric encryption are digital signatures. For digital signatures, the sender encrypts a message with his/her private key. The receiver tries to decrypt the message with the public key of the sender. If the receiver succeeds in the decryption, then this is a proof that both keys belong together. The message really comes from the owner or the public key and not from somebody else.

However, there is one difficulty left. How can a communication partner be sure that a public key really belongs to a definite person and not to somebody else, maybe an attacker?

The solution for this provide certificates. A certificate holds the name and the public key of the sender and is digitally signed by a trusted third party. However, the signature of a trusted party only proves that the certificate has been in the hands of this third party. It does not automatically prove the correctness of the certificate (or any other document). For example, an attacker could have caught the original certificate and simple exchanged the contended public key by its own public key, then sends the certificate to the trustful receiver.

In order to prevent such manipulations, it is possible to apply one-way hash functions. The most popular algorithms for the calculation of such hash functions are MD5 (message digest 5) from the year 1992 and SHA (Secure Hash Algorithm) which has been published in the year 1995. They use hash values of a length of 128 and 160 bit, respectively.

In order to give a digital signature that cannot be manipulated, the author calculates the hash value of the document that is to be signed, encrypts the hash value with his/her private key, and adds the encrypted hash value to the document. The receiver splits and decrypts the added hash value, then calculates the hash value of the message on his/her own. If both values match, the receiver can be sure that the message comes from the signing sender and has not been manipulated.

4. Security Concept

In section 1, we have exposed authentication, authorization, privacy, and integrity as the challenges for a security concept. For the UniCats system as it has been presented in section 2, we see the following four starting points for a security improvement:

- Authentication of customers: The identity of a customer must be assured.
- Authorization of customers: Customers using the system may have different rights. E.g., in a university library system, faculty members may have more rights than students, and these have more rights than external users.
- Authentication of agents: The identity of an agent must be assured.
- Secure communication channels: It must be possible to encrypt messages transmitted between agents so that only the intended receiver of the message is able to understand the message and manipulations of the message can be detected.

Authentication and authorization is managed by customer passports. The customer passport verifies that a specific customer has successfully passed login procedure and confirms the rights of the customer in the use of the system. The customer passport accompanies each query and order of a specific customer. After a customer has left the system, all instances of the passport should have been deleted. If the customers logs in again, a new customer passport is created. In addition to that, each customer passport is equipped with an expiration date and becomes invalid after a period of time.

Customer passports are issued by a new agent type, the Customer Authentication Agent (CAA). During login procedure, a Customer Interface Agent contacts the corresponding Customer Authentication Agent and sends the customer's account name and password (in an encrypted form). The Customer Authentication Agent checks whether the customer is registered in its database and the passport matches. Either it answers with a new issued customer passport, which includes the rights of the customer, or it sends an error message, which causes the Customer Interface Agent to refuse the entrance to the system. There is also the possibility to issue an anonymous guest passport; however, this passport has very limited rights.

The authentication of agents can be established very similar to the authentication of customers. An agent that is about to enter a business relation to another agent can ask for this agent's agent passport. The agent passport holds information about the name, current address, and type of an agent. If an agent moves its location or the expiration date is reached, a new agent passport must be issued. Agent passports are issued by an Agent Authentication Agent (AAA).

Customer passports and agent passports are signed with the digital signature of the issuing agent. A hash value is used to detect manipulations.

Since all communication inside a community is done by direct procedure calls, secure communication is only necessary for external communication, when messages are exchanged by means of web services. For performance reasons, we use symmetric encryption and create the symmetric key just in time using the Diffie Hellman algorithm. The protocol we use is the same as the protocol used for secure communication in SSL, so it is likely that we can provide the same level of security.

For secure communication, the UniCats community has been extended and a new module has been added, the Secure Communication Module. This module offers message encryption and decryption by means of a symmetric key negotiated with the community of the communication partner. When an external community is contacted for the first time, the Secure Communication Module invokes the Diffie Hellman protocol in order to create a new symmetric key. A symmetric key is only valid for a pair of communities and never transmitted through the network.

In order to prevent a man-in-the-middle attack, each community should have a communication certificate. Before the negotiation for a symmetric key starts, both communication partner exchange their communication certificates in order to prove the identity of the other communication partner. In contrast to a passport, the certificate also holds the public key of the owner of the certificate. The private key associated with the community certificate is never transmitted. Unless a community owns a certificate, it can not provide secure communication.

Encrypted communication is not necessary in each case. An agent can decide in each single case whether to apply secure communication or regular communication.

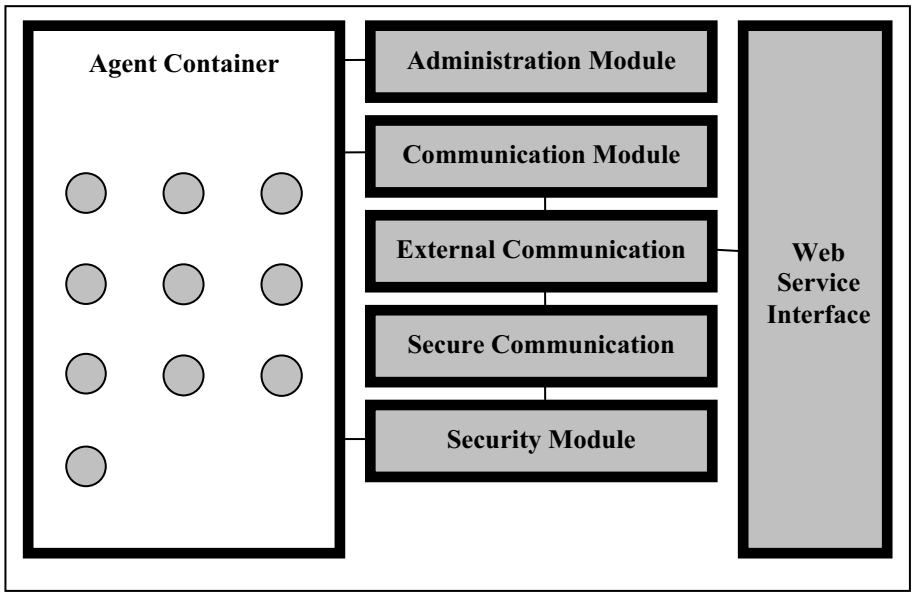


Figure 4. The UniCats community with security extensions

Regular communication is much faster than secure communication and always available.

We have seen that each community needs a community certificate in order to provide secure communication. A similar certificate is needed by the Customer Authentication Agent and the Agent Authentication Agent in order to sign customer and agent passports. These certificates are issued by a new community module, the Security Module. The Security Module holds a permanent and unique certificate, the community certificate. All certificates issued by a Security Module are signed with the community module.

If a community does not own a community certificate, it can ask another community to issue a certificate. This procedure leads to a hierarchy of trusted communities. The private keys associated with the certificates are always kept secret.

Each Security Module holds a list of trusted certificates. Hence, a Security Module is able to prove certificates and passports without having the need to contact the issuer. However, an agent may decide to contact the issuer instead or in addition to proving a passport by the local Security Module, in order to become sure that the passport has not been revoked.

Figure 4 shows the structure of the UniCats community with the two new modules.

5. Realization

The security extensions have been implemented using Java programming language in line with the rest of the UniCats community. However, the security extensions are principally independent from the applied programming language, using only standard protocols and algorithms. For example, secure communication can be established between an agent implemented in pure Java and a second agent implemented in C#/.net.

The new agent types Customer Authentication Agent and Agent Authentication Agent are implemented as extensions of the UniCats standard agent class and are therefore compatible with the rest of the system. Both agent types can be controlled and maintained by an agent control panel.

The Customer Authentication Agent holds an internal database with the registered customers and their passports (in a one-way encrypted form). New customers can be entered by a registration procedure provided by the Customer Interface Agent. Of course, customers can edit their entry, change passports, or delete their registration. Another possibility is to provide a connection to an external database, containing, e.g., a list of all members of a university and their position. The administrator of the agent can edit the registry of the registered customers at any time, delete a customer, whose registration has been revoked, or change the rights of a customer. The Customer Authentication Agent also holds a revocation list, publishing those customers whose registration has been revoked, but might have a passport still valid.

The Agent Authentication Agent is very similar to the Customer Authentication Agent. In order to decide whether a passport is issued for a specific agent, the issuing agent checks whether the other agent has a registered type. The Agent Authentication Agent holds a revocation list, too.

The Security Module issues certificates to the own Secure Communication Module, the two types of authorization agents, and other communities. Issuing a certificate to another community gives this other community the possibility to issue certificates on its own and can be a source for misuse. Because of this, a community needs the confirmation of a human administrator to issue a certificate to another community. In addition to this, each Security Module holds a revocation list of invalided certificates, which is propagated among the trusted communities.

Each certificate saves the whole history of issuers, i.e., not only the community which issued the certificate, but also the community which issued the certificate of the issuer. This eases the proof of a certificate (or a passport) inside the Security Module, because the Security Module can trust all certificates in this sequence as long as it can trust one of the certificates. Moreover, this also works the other way: If a Security Module cannot trust one certificate (e.g., because this certificate has been revoked), it can also invalidate the certificates following in the hierarchy.

For the Security Module, a separate control panel (the security control panel) has been created and integrated in the frame of the administration control panel.

6. Related Work

The most popular protocol for providing secure transmissions through the Internet is the Secure Socket Layer SSL [6]. SSL bases on both certificates and data encryption, using a symmetric key that is created with the Diffie Hellman Algorithm. SSL has two major disadvantages in regard of web services: Each messages transfer has to be encrypted, even if this is not necessary, e.g., if the content of the message does not need to be kept secret. The second disadvantage is that the message is encrypted completely. For a web service application, it would be more appropriate to encrypt only some parts of the messages, while other parts such as the address or transport information should be readable by everyone.

For our security enhancement, we used the same protocols as SSL. Since we implemented the encryption in the application level, we could avoid its disadvantages.

Another important approach in the encryption of XML documents comes from the World Wide Web Consortium W3C [8]. XML Encryption replaces any part of the XML tree by an encrypted element that is enclosed in an EncryptedData tag. XML Signature supports the transport of digital signatures. For a detailed description see [5]. The XML Key Management Specification [W3C-XK] supports the access to a public key infrastructure and provides a mechanism for the registration of public keys independent from the actual implementation of the certificates (compare [7]).

7. Conclusion

In this paper, we have presented the idea and the realization of providing security to a platform for an electronic market in the field of scientific literature. The security enhancement encompasses four objectives: authentication, authorization, privacy, and integrity. The main parts of our solution are a combination of mutual exchange of signed certificates and (lightweight) passports and the encrypted communication on the base of web services.

The main advantage of our approach is the security mechanism may only be used when it is necessary. Secure communication is only applied when private data of a customer or agent are transmitted; an exchange of agent passports or communication certificates only takes place the first time when two agents or communities come into contact and the authentication is mandatory. In general, the decision whether security mechanisms are applied or not is left to the agents.

There are several extensions worth to be considered in the future:

- The type checking used by the Agent Authentication Agents and the Security Module for their decision, whether to issue an agent passport or a certificate for an authentication agent, is not safe. As long as all source code for all agents is freely available, an attacker could simply camouflage his/her own agent with a standard agent type, calculating the necessary hash value on his/her own. A solution for this would be to register all agents by a central authority and include a unique agent identifier in the source code of an agent implementation.

- We have applied authentication and secure communication only inside the UniCats environment. There is no guarantee about the connection to customer and providers. However, these connections are very individual and depend on the preferences of customers and providers. It is not possible to create a general solution.
- In order to have a proof about the business transactions performed within the environment, it would be necessary to have a log of all these transactions. Agents do have a private log, but this could be faked very easily. So there should be a separate trusted log instance, where all transactions are reported. However, since all messages had to be sent twice, such log instance would cause a sincere increase of network traffic. Moreover, monitoring all steps of a customer would also be a contraction to the goal of data privacy.

References

[1] M. Christoffel: *Information Integration as a Matter of Market Agents*. In: Proceedings of the 5th International Conference on Electronic Commerce Research, Montréal, 2002

[2] M. Christoffel, G. Wojke, M. Gensthaler: *How Many Small Libraries Can Be a Large Library*. In: Proceedings of the 5th Russian Conference on Digital Libraries. St. Petersburg, 2003

[3] M. Christoffel, B. Schmitt, S. Pulkowski, P. Lockemann: *Electronic Commerce: The Roadmap for University Libraries and Their Members to Survive in the Information Jungle*. In: ACM SIGMOD Record 27(4), 1998

[4] M. Christoffel, B. Schmitt, S. Pulkowski, P. Lockemann, C. Schütte: *The UniCats Approach: New Management for Books in the Information Market.* In: Proceedings of the International Conference IuK99 – Dynamic Documents, Jena, 1999

[5] M. Mactaggart: *Enabling XML security.* http://www-106.ibm.com/developerworks/xml/library/s-xmlsec.html/index.html

[6] Netscape: *SSL 3.0 Specification*. http://wp.netscape.com/eng/ssl3/

[7] M. Verma: *XML Security: The XML Key Management Specification.* http://www-106.ibm.com/developerworks/xml/library/x-seclay3/

[8] World Wide Web Consortium: *Homepage of the XML Encryption Workgroup*. http://www.w3.org/Encryption/2001/

[9] World Wide Web Consortium: *Homepage of the XML Key Management Specification*. http://www.w3.org/2001/XKMS/