Department of Computers, Czech Technical University, Prague
Department of Software Engineering, Charles University, Prague
Department of Computer Science, VŠB-Technical University of Ostrava
ČSKI, OS Computer Science and Society

Proceedings of the Workshop

DATESO 2003

Databases, Texts, Specifications, and Objects

April 9 – 11, 2003
Desná – Černá Říčka

Technical editor: Jiří Dvorský

VŠB-Technical University of Ostrava

DATESO 2003 (Proceedings of the Workshop Databases, Texts, Specifications, and Objects), April 9-11, 2003, Desná – Černá Říčka, 1st – Ostrava – VŠB-Technical University of Ostrava, Ostrava-Poruba, tř. 17. listopadu 15, 708 33

ISBN 80-248-0330-5

Preface

DATESO 2003, the international workshop on current trends on Databases, Information Retrieval, Algebraic Specification and Object Oriented Programming, was held on April 9th 11th, 2003 in Desná – Černá. This was the third annual workshop organized by FEL ČVUT Praha, Department of Computer Science and Engineering, MFF UK Praha, Department of Software Engineering, and VŠB-Technical University Ostrava, Department of Computer Science. The DATESO aims for strengthening the connection between this various areas of informatics. The proceedings of DATESO 2003 are also available at Web site `http://www.cs.vsb.cz/dateso/2003/`.

We wish to express our sincere thanks to all the authors who submitted papers, the members of the Program Committee, who reviewed them on the basis of originality, technical quality, and presentation. We are also thankful to the Organization Committee.

May, 2003 J. Pokorný, V. Snášel (Eds.)

Program Comittee

Jaroslav Pokorný	Charles University, Prague
Karel Richta	Czech Technical University, Prague
Václav Snášel	VŠB-Technical University of Ostrava, Ostrava
Pavel Zezula	Masaryk University, Brno

Organization Comittee

Yveta Geletičová	VŠB-Technical University of Ostrava
Pavel Moravec	VŠB-Technical University of Ostrava
Aleš Keprt	VŠB-Technical University of Ostrava
Jiří Dvorský	VŠB-Technical University of Ostrava

Table of Contents

Compression-Based Models for Processing of Structured and Semi-structured Data

Abu Sayed Md. Latiful Hoque
Latiful.Hoque@cis.strath.ac.uk

Department of Computer and Information Sciences, University of Strathclyde, 26, Richmond St, Glasgow, G1 1XH, UK

Abstract. Lossless data compression is potentially attractive in database systems for reduction of storage cost and performance improvement. It has proved difficult to combine a good compression technique to achieve both performance improvement and storage reduction. This paper presents a novel three-layer database architecture for storage and querying of structured relational databases, sparsely-populated e-commerce data and semi-structured XML.
We have proved the system in practice with a variety of data. We have achieved significant improvement over the basic Hibase model [28] for relational data. Our system perform better than the Ternary model [22] for the sparsely-populated data. We compared our results with conventional LZW based UNIX utility *compress*. Our system performs a factor of two to six more in reduction of data than *compress*, maintaining the direct addressability of the compressed form of data. Our method needs two passes which is not a problem for database applications.

1 Introduction

Reducing the volume of data without losing any information is known as Lossless Data Compression. This is potentially attractive in database systems for two reasons:

1. reduction of storage cost
2. performance improvement

The reduction of storage cost is obvious. The performance improvement arises because the smaller volume of compressed data may be accommodated in faster memory than its uncompressed counterpart. Only a smaller amount of compressed data need be transferred and/or processed to effect any particular operation. A further significant factor is now arising in the storage and transfer of semi-structured XML data across the Internet and distributed applications using wireless communications. Bandwidth is a performance bottleneck and data transfers may be costly in wireless systems. All of these constitute factors making data compression architecture exceedingly worthwhile.

To achieve the above goals, we have developed a three layer database architecture (Fig. 1) using compression.

© J. Pokorný, V. Snášel (Eds.): Dateso 2003, pp. 1–20, ISBN 80-248-0330-5.
VŠB – Technical University of Ostrava, Dept. of Computer Science, 2003.

Layer 1: The lowest layer is the vector structures to store the compressed form of data. As queries are processed on the compressed form of data, indexing is allowed on the structure such that we can access any element in the compressed form without decompression. The size of the element can vary during database update. The vector can adapt dynamically as data is added incrementally to the database. This dynamic vector structure is the basic building block of the architecture.

Layer 2: The second layer is the explicit representation of the off-line dictionary in compact form. We have presented a phrase selection algorithm for off-line dictionary method in linear time and space [6].

Layer 3: The third layer consists of the data models to represent structured relational data, sparsely-populated data and semi-structured XML.

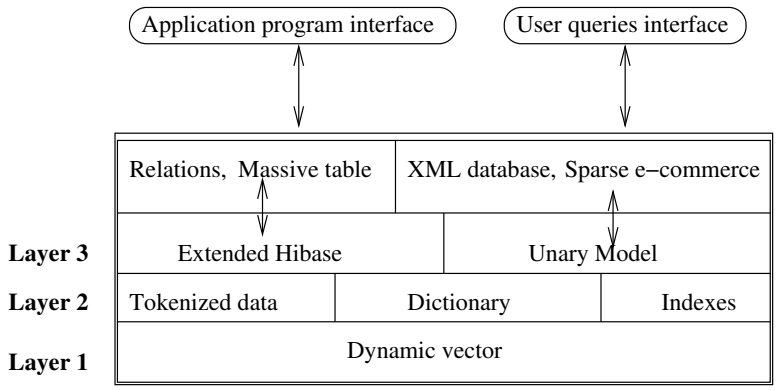


Fig. 1. The three layer database architecture using an off-line dictionary method and the unary model

The Hibase model by Cockshott, McGregor and Wilson [28] is a model of data representations based on information theory. We have developed an extention of the Hibase model for storage and querying of the structured relational databases. Storage and querying sparsely-populated e-commerce data [22], semi-structured data such as XML [7] and the massive table data [25] typically of communication networks has created new challenges for comventional database systems. The horizontal relational model is less suitable for processing of these types of data because of space overheads and performance limitations. We have developed a novel unary model for storage and querying of sparsely-populated data. We have extended the unary model for XML data.

In section 2, a review of the research in compression methods in database systems is presented. We have taken Hibase as the basis of our compression method. We have described the dictionary organization and compressed relational structure of Hibase model and the Extended Hibase model in section 3. Section 4

presents the details of the unary data model. Section 5 describes the extension of unary data model for XML data. Section 6 details the experimental work that has been carried out. The experimental evaluation has been performed using a wide variety of real and synthetic datasets. Section 7 presents conclusions and suggestions for future work.

2 Literature Review

Wee et. al. [29] have proposed a tuple level compression scheme using Augmented Vector Quantization. Vector quantization is a lossy data compression technique used in image and speech coding [3]. However they have developed a loss-less method for database compression to improve performance of I/O intensive operations.

A similar compression scheme has been given in [27] with a different approach. They have presented a set of very simple and light-weight compression techniques and show how a database system can be extended to exploit these compression techniques. Numeric compression is done by suppressing zeros, string compression is done by classical Huffman [13] or LZW [30] and dictionary-based compression methods are used for the field containing a small number of different values.

The scalability of MMDB depends on the size of the memory resident data. Work has already been done on the compact representation of data [21,14]. A typical approach to data representation in a such system is to characterize domain values as a sequence of strings. Tuples are represented as a set of pointers or tokens referring to the corresponding domain values [21]. This approach provides significant compression but leaves each domain value represented by a fixed length pointer/token that would typically be a machine word in length. An alternative strategy is to store a fixed length reference number for each domain value instead of a pointer to the value [21]. The benefit of this approach is that in a static file structure, careful choice of the reference number will provide optimal packing of tuples. Another optimal approach for generating lexical tokens is given by [8].

XML is emerging as a new major standard for representing data on the world wide web. Effort to standardize data exchange over the internet has focussed on the potential of XML. This has led to research on the possible approaches to the representation and use of relational databases for storing XML. Researchers have proposed several transformation models for this purpose. Schmidt et. al. [7] have developed an XML extention of Monet based on the binary relational model [10]. A semantic compression method optimized for XML has been given in [17]. The method work well for data exchange and archiving. However it is not designed for direct querying and achieves optimum performance only when the data sets are large.

The purpose of using compressed text databases is to store both text and indexes in compressed form so that queries can be processed on the compressed data without prior decompression. Manber [19] has developed a dictionary

method based on directed graph $G = (V, E)$, such that the vertices of G correspond to the different characters in the text, one vertex for each unique character and the edges correspond to the character pairs. The frequency count of each pair is the weight of the pair. Finding the best non-overlapping phrase is an $NPcomplete$ problem. A greedy algorithm with limited size of the graph has been used.

The index (or concordance) of a full text retrieval system is one of the largest components of the system. The size of the uncompressed concordance may be 50% to 300% of the size of the uncompressed text [23]. Linoff et. al [18] used a combination of compression methods to reduce the size of the concordance.

Effective exploratory analysis of massive, high-dimensional tables of alphanumeric data is a ubiquitous requirement for a variety of application environments including corporate data warehouses, network traffic monitoring or large socio-economic or demographic serveys [24]. An example of massive data tables is "Call Detail Records" (CDR) of large telecommunication systems. A typical CDR is a fixed length record structure comprising several hundred bytes of data that capture information on various attributes of each call. These CDRs are stored in tables that can grow to truely massive sizes, in the order of several terabytes per year. Table compression was introduced by Buchsbaum et. al. [4] as a unique application of compression, based on several distinguishing characteristics. They have introduced a system called *pzip*. The basis of the method is to construct a compression plan studying a very small training set off-line.

Sparsely-populated data arises when sometimes data is represented using a single horizontal table. An example is the sparse bit-map for a digital library [20], the electronic marketplace in [22] and the new-portal system in IBM-Almaden [2]. In a relational database system, data objects are conventionally stored using a horizontal scheme. A data object is represented as a row of a table. There are as many columns in the table as the number of attributes of the objects. To store all the information in one table for this kind of applications, most of the attribute values become null. Storage and query problems arise using the horizontal relational model. Agrawal et. al. [22] have developed a ternary vertical model for a compact representation of sparsely-populated data.

3 The Extended Hibase Model

3.1 Basis of the Approach

In the relational approach [9], the database is a set of relations. A relation is a set of tuples. A tuple (row) in relation r represents a relationship among a set of values. The corresponding values of each tuple belong to a domain, for which there is a set of permitted values. If the domains are $D_1, D_2, \ldots Dn$ respectively, then relation r is defined as the subset of the Cartesian product of the domains.

Thus r is defined as

$$r \subseteq D_1 \times D_2 \times \ldots \times D_n$$

A relation represents a set of tuples. In a conventional table structure, the rows represents tuples and the columns contain values drawn from domains.

Queries are answered by the application of the operations of the relational algebra, usually as embodied in a relational calculus-based language such as SQL.

Table 1. 'CUSTOMER': an example of a simple relation .

Customer name	Street	City	Status
Billy	Albert	Glasgow	Married
Ann	Albert	Edinburgh	Single
John	North Hover	Glasgow	Married
Annan	Maxwell	Glasgow	Married
Johnes	Albert	Edinburgh	Married
Billal	Albert	Glasgow	Married
Peter	Maxwell	Edinburgh	Married
John	Maxwell	Glasgow	Single

3.2 Compression Architecture: HIBASE Approach

The design philosophy of the compression architecture is to aim for a cost/performance between that of conventional DBMS and main memory DBMS - costs less than the latter, and processes faster than the former [28]. The architecture's compact representation can be derived from a conventional record structure in the following steps:

Create a Domain Dictionary A dictionary per domain is employed to store string values and to provide integer identifiers for them. This achieves a lower range of identifiers, and hence a more compact representation than could be achieved if only a single dictionary was provided for the whole database.

Replace the original field values of the relation by identifiers The range of the identifiers need only be sufficient to unambiguously distinguish which string of the domain dictionary is indicated. For instance, in the example shown in figure 2 since there are only 7 distinct customer names, only seven identifiers 0-6 inclusive are required. This range can be represented by only a 3-bit binary number. Hence in the compressed table each tuple requires only (3 bits : Customer name) (2 bits: Street) (1 bit: City) (1 bits: Status) - a total of 7 bits instead of the 29 *8 bits for the uncompressed relation. This gives a compression of the table itself by a factor of over 33. This is not the overall compression ratio however because we must also take account of the space occupied by the domain dictionaries and also by indexes. Typically a proportion of domains are

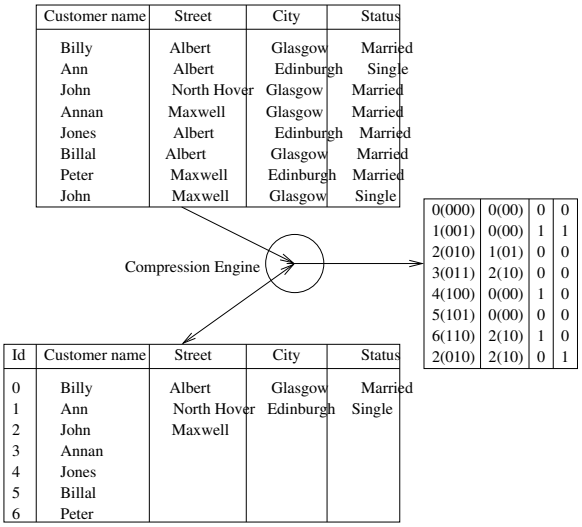


Fig. 2. Compression of a relation using domain dictionaries

present in several relations and this reduces the dictionary overhead by sharing it between different attributes.

The fact that in a domain a specific identifier always refers to the same field value enables many operations to be carried out by processing only the table component of the data, ignoring the dictionaries until string values are essential (e.g. for output).

Dictionary Structure All the unique attribute values (lexemes) are stored in an end-to-end format in a string heap. A hashing mechanism is used to achieve a contiguous integer identifier for the lexemes. This reduces the size of the compressed table. It has three important characteristics:

1. It maps the attribute values to their encoded representation during the compression operation: $encode(lexeme) \longrightarrow token$.

2. It performs the reverse mapping from codes to literal values when parts of the relation are decompressed: $decode(token) \longrightarrow lexeme$.

3. The mapping is cyclic such that lexeme = decode(encode(lexeme)) and also token = encode(decode(token)).

The structure is attractive for low cardinality data. For high cardinality and primary key data, the size of the string heap grows considerably and contributes very little or no compression.

Processing in the compressed data phase Queries are carried out directly on the compressed data, without requiring any decompression during the processing (figure 3). The query is translated to compressed form and then processed

directly against the compressed form of the relational data. Less data needs to be manipulated and this is more efficient than the conventional alternative of processing an uncompressed query, against uncompressed data. The use of this compression method also increases efficiency of buffering of data both in RAM and in CPU caches.

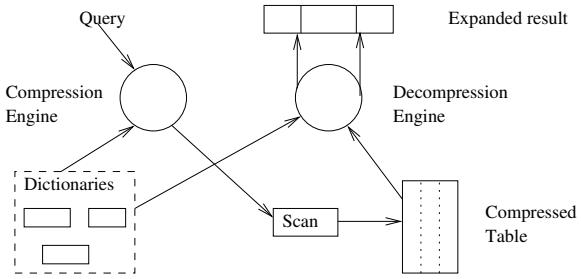


Fig. 3. Querying a database in compressed form

The final answer has to be converted from the compressed form to a normal uncompressed representation. However, the computational cost of this decompression is low because the amount of data that needs to be decompressed is only a small fraction of that processed.

Column-wise representation of relations The architecture represents a table in storage as a set of columns, not a set of rows. This makes certain operations on the compressed database considerably more efficient. Of course, the user is free to regard the table as a set of rows as he/she pleases.

Representation by column can be processed more efficiently than representation by row. A column-wise organisation is much more efficient for dynamic update of the compressed representation. A general database system must support dynamic incremental update, while maintaining efficiency of access.

The processing speed of a query is enhanced because, typically, queries specify operations on only a subset of domains. In a column-wise database only those specified values need be transferred, stored and processed. This requires only a fraction of the data volumes required when processing by rows.

3.3 The Extended Hibase Model

We have developed the more compact and efficient vector structure as given in [5]. The second order compression using an off-line dictionary method ofer a further compression of the dictionary heap in Hibase. This is described in [6]. The use of the improved vector structure and second order compression of the dictionary heap creates the extended Hibase model.

The extended Hibase model can achieve some benefit in a very low cardinality data using unary representation. For example, if a domain has three values: "Yes", "No" and "undefined", it requires two bits to represent three values. If there are 15 consecutive "yes" value, it requires 30 bits. The unary representation can reduce it to only 6 bits.

The single attribute index and the composite index [28] are created on the compressed representations in the Extended Hibase model using a hashing mechanism. The Update operation in the Extended Hibase model requires a 'string' look up in the dictionary. If the 'string' is present in the dictionary the Update operation does not need a reorganization of the vector structure. If the 'string' is not present in the dictionary it is inserted into the dictionary. This insertion might result an increase of the element size. In this case the Update operation requires a reorganization of the vector structure. This reorganization is less costly in the Extended Hibase model than the basic Hibase model. Deletion is performed by putting a null representation in the corresponding element position. Dictionary entries are not deleted.

4 The Unary Model

We have developed a unary model for representing sparse data. It overcomes the space difficulties of both the previous methods (Horizontal model (Table 2) and Ternary representation [22]). As will be seen, it also offers better performance. The unary model creates one unary compressed table for each attribute maintaining the record identifier implicitly. The unary tables is shown in table 3.

Table 2. A sparsely-populated news portal table.

News Object	Country	People	Sport	Research
news1	UK	⊥	Atheletic	⊥
news2	⊥	McGregor	⊥	Database
news3	USA	⊥	⊥	⊥
news4	⊥	⊥	Football	⊥
news5	⊥	Newton	⊥	Physics

Compressed representations The unary representation is effectively an encoding of the off-line dictionary method [6] and a run-length encoding [11] of the uncompressed sparse-table. Using the off-line approach [6], five dictionaries were created for five domains: *News identifier, country, people, sport and research.* The domain dictionaries are given in figure 4.

We have developed two alternative compressed representations for unary tables.

Table 3. Unary table.

Country
UK
⊥
USA
⊥
⊥

People
⊥
McGregor
⊥
⊥
Newton

Sport
Atheletic
⊥
⊥
Football
⊥

Research
⊥
Database
⊥
⊥
Physics

1. The offset structure
2. The bit-array index structure

An offset representation of unary tables shown in Table 3 is created using run length encoding [11]. The run length of each non-null value is stored as an offset of that value. An element of the compressed unary table is the union of the unary table offset and the dictionary identifier. For example, the unary table offset of the first non-null value "McGregor" of *people* domain is "1" and the dictionary identifier is "0". The element in the unary compressed representation is "10". The compressed unary tables are shown in Figure 5.

The bit-array representation is shown in Figure 6. In this representation a bit-array of length equal to the cardinality of the sparse table is used. In the bit-array a '1' represence of an attribute value and a '0' represents 'null'.

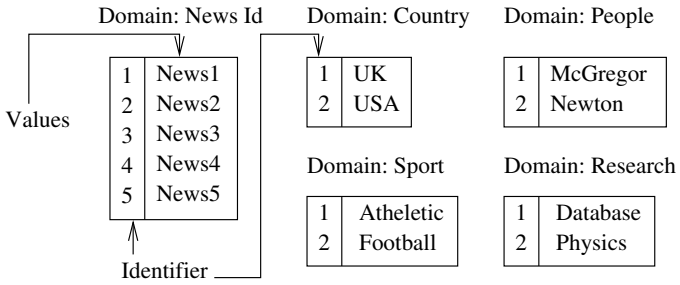


Fig. 4. Domain dictionaries for the unary representations

Let n be the cardinality of the horizontal sparse table and m be the number of columns. The non-null density is ρ and the density is uniformly distributed. The average run-length between non-null values is d. The number of additional bits for the run-length is $mn\rho \log_2(d+1)$ bits, and $0 \leq d \leq (n-1)$. For uniform distribution, $d = (1-\rho)/\rho$, where $1/n \leq \rho \leq 1$. The size of the compressed table using offset is $mn\rho \log_2(1/\rho) + mn\rho c$. When $\rho = 1$, the size of the offset method

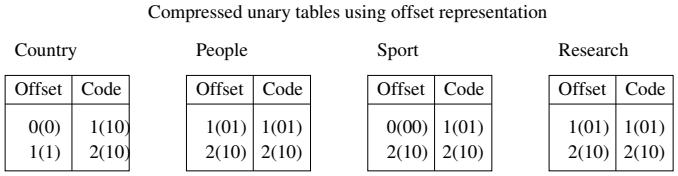


Fig. 5. Compressed representations of unary tables using offset

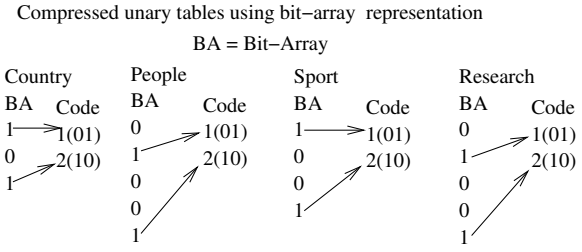


Fig. 6. Compressed representations of unary tables using bit-array index

approaches to the size of the compressed horizontal table. The size of the bit-array index is mn bits and the total size of the unary table is $mn + mn\rho c$.

5 Unary Model in XML

To fully realize the potential of XML as a basis for data storage and communication on the Internet, the performance issues inharent in the language must be resolved. An XML document can be represented using a syntax tree [7]. Starting from the syntax tree representation, we have developed a compressed representation of XML data using unary model.

The method given in [7] converts XML data into a binary relational model. The number of tables are equal to the number of edges in the syntax tree that underlies the XML structure. Each name of the table signifies for which edge the table is created. The transformation of XML data into binary relational model has two main problems. One is the space overhead and the other is the large number of tables generated. An $n - ary$ query on the data requires an $n - ary$ join. Alternatively, XParent [12] creates four tables to represent XML data but has high data redundancy.

We can represent XML data in compressed form using unary model described in the previous section. We have shown that the model can process data on the compressed form more efficiently than in the uncompressed form. Indexing is possible in the compressed form as well. We have transformed XML data into the unary model using Monet transformation [7]. A binary tuple was created

for each edge of the syntax tree. The number of unary tables were equal to the number of nodes in the tree.

For example, an XML entry for a news portal database is given in figure 7. Exploring the structures of the document, we can represent it by syntax a tree. In the example, news portal is the root of the tree and other attributes are the nodes of the tree. The corresponding syntax tree of the document is given in figure 8. The lowest level attributes of the tree point to the data values of those attributes.

The horizontal sparse table is created for the syntax tree using Monet transformation [7]. The sparse table is transformed into unary compressed representation and queries are processed on the compressed unary model.

```
<News Portal>
    <Article key = "C000">
        <Type> Sports  <\Type>
        <Author>Waseem </Author>
        <Title>Cricket</Title>
    <Article>
    <Article key = "S000">
        <Editor>Millar</Editor>
        <Author>Waseem </Author>
        <Type>Sports </Type>
        <Country>UK </Country>
     <Article>
<\News Portal>
```

Fig. 7. XML document describing a fragment of news portal database

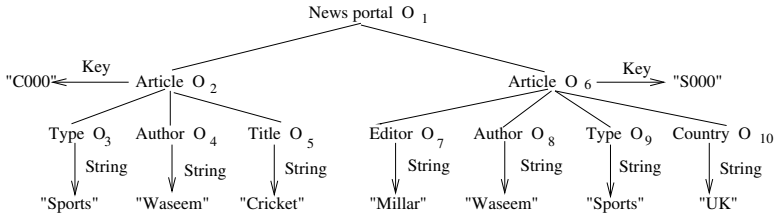


Fig. 8. Syntax tree for the news portal document

All distinct nodes of the same level in the tree are attributes of the horizontal table. The nodes that point only to values are considered as CDATA nodes. Those nodes that have the key values are considered as key nodes. We can consider a single domain or multiple domains for CDATA depending on the

individual attributes and their cardinalities. The syntax tree in figure 8 has been converted to sparse table as given in Table 4.

We have considered one column for the root and a column for every child. We have the following attributes: *News Portal, n.Article, n.a.Type, n.a.Author, n.a.Title, n.a.Editor, n.a.Country, n.a.key, n.a.CDATA* for the sparse table. We have considered a binary tuple for each of the edge of the tree. If there are n number of attributes in the sparse table, for each tuple in the horizontal table, $(n-2)$ *null* values are added. The non-null density ρ will be $2/n$, where n depends on the depth of the tree. For database applications, the value of n is quite high, and hence the unary model offers better performance in all the ranges of non-null densities.

Table 4. Horizontal sparse table

News	n.Article	n.a.Type	n.a.Author	n.a.Title	n.a.Editor	n.a.Country	n.a.Key	n.a.Cdata
O_1	O_2	null	null	null	null	null	null	null
null	O_2	null	null	null	null	null	C000	null
null	O_2	O_3	null	null	null	null	null	null
null	null	O_3	null	null	null	null	null	Sports
null	O_2	null	O_4	null	null	null	null	null
null	null	null	O_4	null	null	null	null	Waseem
null	O_2	null	null	O_5	null	null	null	null
null	null	null	null	O_5	null	null	null	Cricket
O_1	O_6	null	null	null	null	null	null	null
null	O_6	null	null	null	O_7	null	null	null
null	null	null	null	null	O_7	null	null	Millar
null	O_6	null	O_8	null	null	null	null	null
null	null	null	O_8	null	null	null	null	Waseem
null	O_6	O_9	null	null	null	null	null	null
null	null	O_9	null	null	null	null	null	Sports
null	O_6	null	null	null	null	O_{10}	null	null
null	null	null	null	null	null	O_{10}	null	UK
null	O_6	null	null	null	null	null	S000	null

6 Results and Discussions

All experiments were run on an 800 MHz AMD DuronTM processor machine with 256 MB of physical memory. The operating system was Microsoft Widows 2000 Professional. We have implemented the model using Sun JDK version 1.3. Both real (Scittish Judges, River [1], Test1, Test2 and Internet) and synthetic (Sparsely-populated) datasets were used in the experiments.

We have evaluated the effect of storing a single domain dictionary and one dictionary per distinct columns. We can use a single domain dictionary as is

the case in ternary model [22]. All the data values have been considered as VARCHAR data. The choice of domain strategy has a significant effect on overall compression. The effect of the number of domain dictionaries on compression is shown in Figure 9.

The overall size of the dictionary using a single dictionary is always less than the same for multiple domains (Figure 10). The overall compression is dependent on the sum of the dictionary and the compressed representation. The length of the token is greater in the single dictionary approach than in the multiple dictionary approach. This is why the overall compression using the multiple dictionary approach is always higher than the single dictionary approach (Figure 9).

We have achieved three main advances over the Hibase approach:

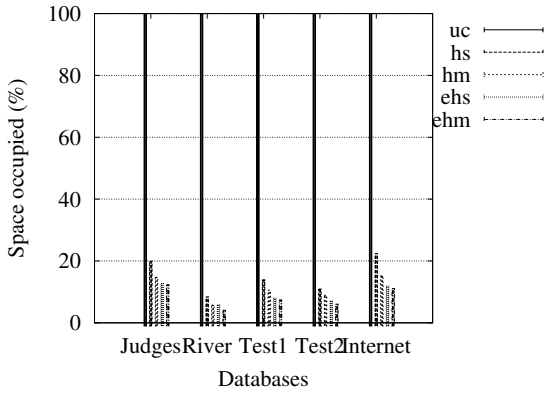
1. The improvement by using improved vector structure
2. The improvement by using off-line dictionary method
3. The improvement by the unary model

We have achieved in an average a factor of 15 compression for relational databases whereas the basic Hibase approach can achieve a factor of 10 compression (Figure 9). The improved vector structure and the off-line dictionary have the main contribution on the preformance improvement over Hibase (Table 5). Unary model has significantly less impact in this case. The best performance is the use of extended Hibase with multiple dictionaries. The total size of the dictionaries in multiple dictionary option is greater than the corresponding size of the dictionary in single dictionary option (Figure 10). As the size of the token become shorter in multiple dictionary case, the sum of the compressed representation and the dictionary is smaller than the other options.

The selection of the number of domains has more impact on compression in the basic Hibase model than the same for the extended Hibase model. This is because of that the main contribution in extended Hibase model comes from the vector, not the dictionaries (Figure 9).

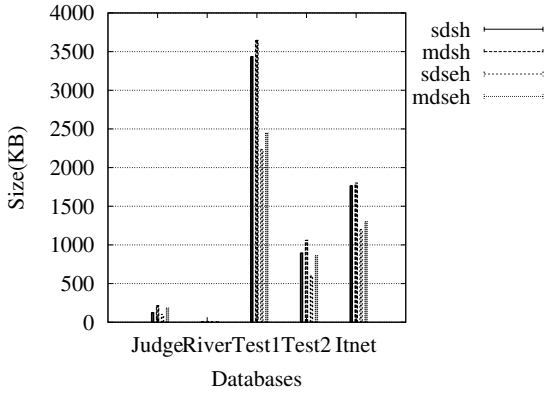
Table 5. The contributions of improved vectors, off-line method and unary representation on performance improvement in Extended Hibase for relational data.

Dataset	Hibase Size (MB)	Extended Hibase Size (MB)	Perfor improvement	Contribution of vector	Contribution of off-line method	Contribution of unary representation
Judges	0.968	0.808	0.20	0.1423	0.04	0.0176
River	0.174	0.123	0.41	0.282	0.00	0.1278
Test1	12.56	8.812	0.425	0.289	0.136	0.00
Test2	6.105	4.14	0.47	0.42	0.048	0.00
Internet	4.6976	3.44	0.3656	0.2202	0.145	0.0163



uc = Uncompressed, hs = Hibase Single Dictionary
hm = Hibase Multiple Dictionary
ehs = Extended Hibase Single Dictionary
ehm = Extended Hibase Multiple Dictionary

Fig. 9. The space occupied in Hibase and Extended Hibase as % of uncompressed size (In assence these are in order from top to bottom and left to right).



sdsh = Single Dictionary Size in Hibase
mdsh = Multiple Dictionary Size in Hibase
sdseh = Single Dictionary Size in Extended Hibase
mdseh = Multiple Dictionary Size in Extended Hibase

Fig. 10. The sizes of single and multiple domain dictionaries (In assence these are these are in order from top to bottom and left to right).

The ternary table grows high as non-null density increases. Figure 11 shows a comparison of the size of the table in ternary and unary representations. The growth in ternary representation is significantly higher than the unary form. The size of the table in horizontal relational storage representation is 1440 MB. We have considered the non-null density ranges from 1% to 30%. For 1% non-null density, the unary representation using offset is 1.5 MB. So the unary offset representation is a factor of 960 smaller than the horizontal sparse table. This high gain is because the horizontal representation has 99% of null values whereas, unary representation has no null values. The ternary representation has no null values. In 30% non-null density, the size of the ternary representation is 887 MB. The same is using unary offset representation is 73.6 MB. So the unary representation is a factor of 20 smaller than the corresponding ternary form. The main contribution in compression is using the unary model (Table 6). The contributions of the vector and the off-line method are insignificant. We have found that the same is true for all range of non-null densities. This is because of the data redundancy in ternary representation. The query processing using the unary representation is also significantly faster (Figure 12).

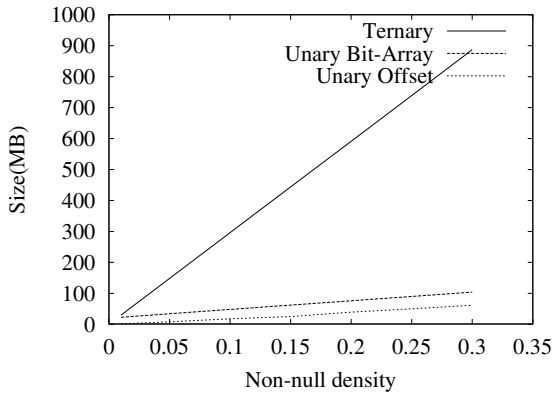


Fig. 11. Comparison of the sizes unary and ternary representations for different non-null densities

UNIX *compress* is a standard compression system using LZW [30] method. We have compared the unary compressed representation with *compress*. Figure 13 shows the comparison of unary model with *compress*. The same comparison has been given for XML data in Figure 14. The unary model performs a factor of 2 - 6 better than the *compress* compression. This is because of the off-line selection of phrases and the compressed unary representation. Though the optimal selection of phrases is an $NP-Complete$ problem [26], we have a near optimal selection by the vertical scanning of the domain and second-order compression described in [6].

Table 6. The contributions of improved vectors, off-line method and unary representation on performance improvement of the Unary model for sparsely-populated data.

non-null densities	Ternary size (MB)	Unary offset size (MB)	Performance improvement	Contribution of vector	Contribution of off-line method	Contribution of unary representation
1 %	30	1.5	20	0.2	0.1	19.7
5 %	148	8.075	18.32	0.201	0.062	18.057
10 %	296	17.9	16.53	0.196	0.039	16.295
15 %	444	25.05	17.72	0.21	0.032	17.478
20 %	591	39.2	15.07	0.191	0.023	14.856
25 %	739	50.0	14.78	0.2	0.018	14.562
30 %	887	61.6	14.4	0.195	0.016	14.189

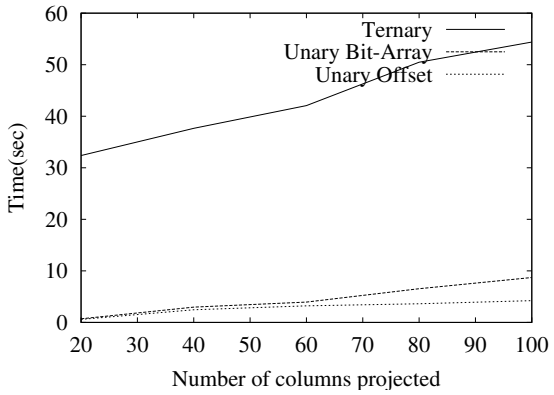


Fig. 12. The projection time for varying number of columns with 5% non-null density

In databases, the data redundancy arises within the attribute values. It is rare to occur redundancy across the attribute values. The conventional dictionary methods [15,16,30] scan the data horizontally and the initial part of the dictionary is very inefficient. As dictionary grows bigger, longer strings are replaced by the codes and the selection of phrases goes towards optimal. So these methods are asymptotically optimal [26]. That is why our method performs always better than the conventional methods. We can compare our results with XMill [17], a semantic compressor for XML data. XMill compresses XML data to a maximum factor of two better than comventional compression methods [15,16,30]. Our system perform significantly better than XMill maintaining the direct addressability of data.

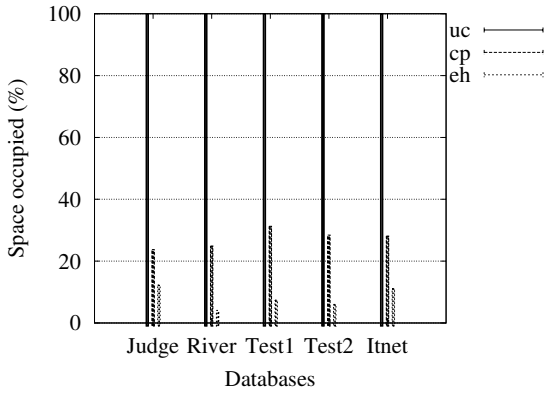


Fig. 13. The comparison of the Extended Hibase model with the UNIX *compress* for relational datasets.

7 Conclusions and Further Research

The performance gain of the Extended Hibase model over the basic Hibase model is a factor of 1.5 in terms of space(Fig. 9). We have also evaluated the individual contributions of the dynamic vector structure, the off-line method and the unary model()Tab. 5). In Extended Hibase model the contribution of the improved vector structure is more than 70% of the total improvement for all the datasets except Internet data (61%). The contribution of unary model for all the datasets is neglegible except the River data (29% of the total improvement).

We have compared our results with the Ternary representation [22] for the sparsely-populated data. The performance improvement in terms of space ranges

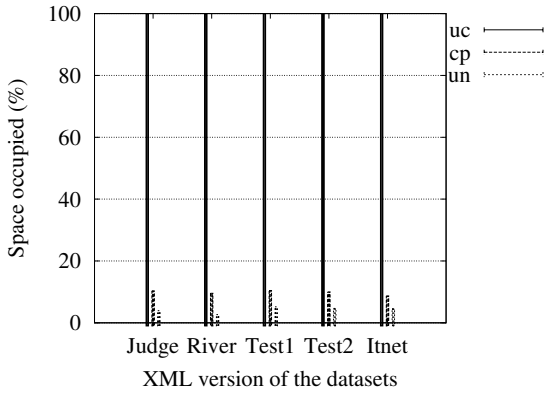


Fig. 14. The comparison of the Unary model with the UNIX *compress* for XML data.

from a factor of 14 to 20 compared to the Ternary model (Fig. 11). We have also evaluated the individual components of the architecture. More than 98% of the total performance gain comes from the unary model (Tab. 6). The query performance of unary model is significantly better than the Ternary model 12.

We have considered our system as a "compressed and memory resident database" paradigm that operates within a single user environment. The future work of this research could be to explore the potential of the architecture to the following other areas:

• The use of a block-oriented vector to allow the extension of the system to distributed multi-user environment. Vertical and horizontal fragmentation can be applied to the compressed representation.

• If the size of the compressed database grows to such an extent that it cannot fit into memory, the same architecture can be extended for disk resident database systems. In that case, data may be partitioned into frequently used and rarely used fragments. The frequently used data can be memory resident and the rarely used data can be disk-resident. Recovery in the compressed representation needs to be addressed.

References

1. http://enterprise.canberra.edu.au/www/riversurvey.nsf?opendatabase.
2. http://www.almaden.ibm.com/.
3. V. Cuperman A. Gersho. Vector quantization: A pattern-matching technique for speech coding. *In IEEE Communication Magazine*, Vol. 21:15–21, December 1983.

4. K. W. Church G. S. Fowler A. L. Buchsbaum, D. F. Caldwell and S. Muthukrishnan. Engineering the compression of massive tables: An experimental approach. In *Proceedings of the* 11^{th} *Annual ACM-Siam SODA*, pages 175–184. ACM, 2000.
5. D. R. McGregor A. S. M. L. Hoque. Improved compressed data representation for computational intelligence systems. In *UKCI-01*, Edinburgh, UK, September 2001.
6. J. Wilson A. S. M. L. Hoque, D. R. McGregor. Database compression using an off-line dictionary method. *ADVIS, LNCS*, 2457:11–20, October 2002.
7. M. Windhouwer F. Waas A. Schmidt, M. Kersten. Efficient relational storage and retrieval of xml documents. In *Lecture Notes on Computer Science*, pages 137–50. Springer-Verlag, 2000.
8. A. Cannane and H. E. Williams. A compression scheme for large databases. In *Proceedings of Australian Database Conference*, University of Western Australia, Curtin University, Murdoch University and Edith Cowan University, Perth, 1998. Springer.
9. E. F. Codd. A relational model of data for large shared data banks. *In Communications of the ACM*, 13:6:377–387, 1970.
10. G. P. Copeland and S. Khoshafian. A decomposition storage model. In *Proceedings of the 1985 ACM SIGMOD*, Austin, Texas, May 1985.
11. S. W. Golomb. Run-length encodings. *In IEEE Transaction on Information Theory*, (IT-12(3):399-401), 1966.
12. W. Wang H. Jiang, H. Lu and J. X. Yu. Path materialization revisited: An efficient storage model for xml data. In *Proceedings of the Second Australian Institute of Computer Ethics Conference*, Canberra, 2000. Australian Computer Society.
13. D. A. Huffman. A method for the construction of minimum-redundancy code. *In Proceedings of IRE*, 40:9:1098–1101, 1952.
14. J. S. Karlsson and M. L. Kersten. W-storage: A self organizing multi-attribute storage technique for very large main memories. In *Proceedings of Australian Database Conference*. IEEE, 1998.
15. A. Lampel and J. Ziv. A universal algorithm for sequential data compression. *In IEEE Transaction on Information Theory*, (Vol. 23):337 – 343, 1977.
16. A. Lampel and J. Ziv. Compression of individual sequences via variable-rate coding. *In IEEE Transaction on Information Theory*, (Vol. 24):530 – 536, 1978.
17. H. Liefke and D. Suciu. Xmill: an efficient compressor for xml data. In *Proceedings of the Management of Data, Dallas, TX USA*, pages 153–164. ACM, 2000.
18. G. Linoff and C. Stanfill. Compression of indexes with full positional information in very large text databases. In *Proceedings of the ACM SIGIR*, pages 88–95, Pittsburgh, USA, 1993.
19. U. Manber. A text compression scheme that allows fast searching directly in the compressed file. *In ACM Transaction On Information Systems*, 15:2:124–136, 1997.
20. A. Moffat and J. Zobel. Parameterised compression for sparse bitmap. In *Proceedings of the* 15^{th} *Annual International SIGIR 92*, pages 274–285. ACM, 1992.
21. J. Thevenin P. Pucheral and P. Valduriez. Efficient main memory data management using dbgraph storage. In *Prodeedings of the 16th Very Large Database Conference*, pages 683 – 695, Brisbane, Queensland, Australia, 1990.
22. Y. Xu R. Agrawal, A. Somani. Storage and querying of e-commerce data. In *Proceedings of the 27th VLDB Conference*, pages 149–158, Roma, Italy, 2001.
23. H. Roger. Special purpose processors for text retrieval. *Database Engineering*, Vol. 4(No. 1):16–29, 1982.

24. M. Garafalakis S. Babu and R. Rastogi. *SPARTAN:* using constrained models for garunteed-error semantic compression. *SIGKDD Exploration Newsletter*, Vol. 4(No. 1), June 2002.
25. M. Garofalakis S. Babu and R. Rastogi. *spartan* : a modelbased semantic compression system for massive data tables. In *Proceedings of the SIGMOD, Santa Barbara, California, USA*, pages 283–294. ACM, 2001.
26. J. A. Storer and T. G. Szymanski. Data compression via textual substitution. *In the Journal of the ACM*, Vol. 29:928–951, 1982.
27. S. Helmer T. Westmann, D. Kossmann and G. Moerkotte. The implementation and performance of compressed databases. *In SIGMOD Record*, 29:3:55–67, 2000.
28. D. McGregor W. P. Cockshott and J. Wilson. High-performance operations using a compressed architecture. *The Computer Journal*, 41:5:283–296, 1998.
29. K. N. Wee and C. V. Ravishankar. Relational database compression using augmented vector quantization. In *Proceedings of the 11th International Conference on Data Engineering*, pages 540–550, Taipei, Taiwan, 1995. IEEE.
30. T.A Welch. A technique for high-performance data compression. *In IEEE Computer*, 17:6:8–19, 1984.

Data Transfer Between Relational Databases Using XML

Pavel Loupal

Czech Technical University in Prague,
Department of Computer Science and Engineering

Abstract. This paper is concerning data transfer between relational databases using XML format. It contains description of XML interfaces of wide spread relational database systems Oracle and Sybase. A simple data synchronization example shows basic issues of existing database interfaces and a way to process different XML formats using a XSL transformation.
Paper should give a basic overview of these technologies including their drawbacks and show how to use them for data transfers.

1 Introduction

The XML language [4] has been created in 1998 and became a standard format for various data transfers. The advantage of using such kind of transfer media brings some new features. This standard is widely acceptable and there are already plenty of applications able to read and write so formatted files. The standard also calculates with a native support for any international characters set which helps to assist creating language independent solutions.

Concept of relational databases is a bit older (first commercial version of Oracle was released in 1980) but with expansion of XML language their producers implemented a support of this format into their products. This maintenance consists of two basic parts - input and output XML interfaces allowing import and export data in this format and inner handling of XML data e.g. creating native data types storing parsed instances of XML data or extending BLOB capabilities of their system. In context of this paper we will concentrate chiefly on XML interfaces.

Relational databases are widely used for storing data in both industry and non-commercial areas. For many reasons, typically historical and organization matters, they cannot be easily replaced by different technological solutions but the stored data must be accessible and integrated with recent systems. Therefore it is no surprise that support for XML technology becomes very important.

Why to use XML technology instead of existing data manipulation techniques? The "classical" approach is based on application programming interfaces shipped with database products, e.g. C++ APIs. There had been no standardization of such interfaces that makes programmers' work a bit difficult - they had to know and understand many different APIs. Accession of ODBC and JDBC

© J. Pokorný, V. Snášel (Eds.): Dateso 2003, pp. 21–29, ISBN 80-248-0330-5.
VŠB – Technical University of Ostrava, Dept. of Computer Science, 2003.

interfaces has solved these problems but another one stayed - applications cannot be extended without rewriting their source code. Using text-based XML format in conjunction of XSLT transformation allows changing the input (or output) much more easily.

Another advantage of this format is its wide support - data can be processed in any time (because of its known plain format) - by any third-party applications during transfer.

Sections 2, 3 and 4 of this paper describe common concept of data transfer between relational databases, mapping between relational structures and XML format and possible ways of data validation. Section 5 outlines features of existing database XML interfaces and finally section 6 shows a concrete example of data synchronization between two databases.

2 Data Transfer Concept

The usual concept for data transfer using XML is shown in figure 1. We will focus on offline transfers but in general the concepts should be similar for on-the-fly processing as well. Let's suppose two relational database systems with existing XML interfaces. These interfaces are usually parts of those databases. Source data are fetched with XML interface module and mapped to database-specific XML format using specified rules. Any "third party" application e.g. cryptographic module or a simple application for logging traffic on this channel can then process this data.

It is useful to have the XML data exchanged in common format - this makes its treatment easier. In case that the data would differ application should be prepared to work with different data format what makes it worthlessly complicated. Such a transformation can be simply done with a XSL stylesheet (see section 6).

In some cases the transformation would have to be solved by application logic instead of a XSL transformation because of its complicacy. This would make this process a bit cumbersome. We would loose the simplicity of adjustment of a stylesheet. In addition to this there are many extension libraries that add new features to standard XSL functionality such as the XLST extension library [6].

We could of course allow only one way data transfers between a relational database and any other data source (e.g. a structured file) but in our case we can demonstrate both imports and exports from/to a database in one example.

3 RDB - XML Mapping

One of the basic tasks for our case is to transform relational data into a XML instance. This process is called RDB-XML mapping. Usually the data is stored using name of an attribute as a tag (a XML document tree node) name and its value as node value as described in experimental part of this paper in section 6. Every document following this model contains a sequence of "row" elements with nested subelements containing attributes values - this model corresponds

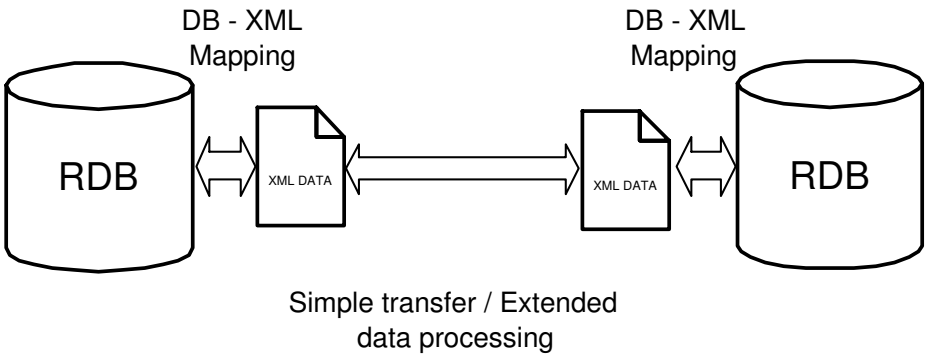


Fig. 1. Data transfer concept

tightly with structure of a table (or view). These examples of straightforward Oracle and Sybase implementations are shown below.

More sophisticated methods use different structure of a document, e.g. XML-DBMS library [2] views an XML document as a tree of objects and then uses an object-relational mapping to map these objects to a relational database. In this view, element types generally correspond to classes and attributes and PCDATA correspond to properties. Complete description and Java implementation could be found at [2].

4 Data Validation

Important feature at application layer is an ability to check data content. Usually we suppose that exported data from database are without faults. In opposite direction it could be useful to have a possibility to validate incoming data in order to prevent data mismatch causing database errors. For XML formatted data there are two possible ways - Document Type Definition (DTD) and XML Schema. These both are W3C consortium [3] standards.

DTD [5] is a part of the XML standard and therefore also the oldest way for checking documents validity. DTD declarations can be defined within the XML document or defined external to the XML document and referenced by it. A DTD defines structure and content of a document. Unfortunately it has also two important drawbacks - its syntax differs from the XML language and there is no way how to define a data type for an element. Everything is simply treated as a text. In spite of this DTDs are sufficient way for simple data validation.

XML Schema [8] standard sprang up to cover needs for detailed checking of document's content. Its syntax is XML compliant which makes it more "readable" for developers. The standard contains strong data typing and structure definition with namespaces support.

XML Schemas are today the most common way for validating data. It has quickly become a widely acceptable standard with built-in support in all significant XML parsers. For instance, Oracle XDK (see following section) can generate a XML Schema depending on a SQL query.

5 XML Interfaces of Relational Databases

This section gives a basic overview of XML interfaces used in two well-known commercial database products - Oracle 8i and Sybase ASE 12.5. - which could be marked as ones of the most significant and technologically advanced companies in that area.

These XML interfaces are the main part responsible for converting relational data into XML documents and conversely. Fetching data is straightforward. A SQL SELECT statement is performed in a source relational database and the data is pulled out in a XML format. There is no standard (DTD or XML schema) saying how should this data be structured.

Importing data into a relational database brings some new issues. XML data files acceptable by database's XML interface are going to be inserted/updated (or deleted from) to a database. Input files are sequentially read and data for single rows are processed. This could bring troubles concerning referential integrity. If a schema has some foreign keys and data transfer starts with "wrong" table, this problem occurs - the mandatory data is not yet in its place in database. One solution how to avoid this kind of problems is to add some application logic to divide this XML document to set of documents containing only one row and control the order of statements to ensure data consistency. This means that there must be a new application layer over database's XML interface. None of databases interfaces described in this paper is so "clever" to check such kind of dependencies. Another solution is to switch off database's referencial integrity before transfer and then start it up again but that is in almost all cases unacceptable.

Let us suppose a simple conceptual database schema shown in figure 2. It is a fragment of a database storing information about books and their authors. Using this example we can describe interfaces' capabilities and their default XML formats.

Oracle. Since Oracle Database version 8i there is a XML module included - XDK (XML Development Kit) [10]. Basic parts of this module are XML parser, XSLT processor and XML-SQL Utility (XSU). All parts are implemented in C++, Java and PL/SQL.

XSU is the most interesting part in context of this paper. It allows creating a XML document depending on a SQL query, importing XML documents in specified format and also dynamically creating a DTD or a XML Schema for data to be exported. That is the only way how to get metainformation about data contained in output XML documents.

These metainformation are evidently taken from system data dictionary and can be used for basic data validation as described in section 4. Some problems

Fig. 2. A simple conceptual schema

could occur when working with constrains (e.g. `CHECK (pages > 100)`) which is not covered by this schema. In this case an external schema must be created.

Let's suppose a SQL query `SELECT ISBN, Title, Price FROM Book`. Using the Oracle interface (method)we get a document shown in figure 3. Such a file consists of tags for rows and nested tags for their attributes.

```
<?xml version = '1.0' encoding = 'ISO-8859-2'?>
<ROWSET>
  <ROW num="1">
    <ISBN>80-85615-74-8</ISBN>
    <TITLE>LaTeX for Beginners</TITLE>
    <PRICE>129</PRICE>
  </ROW>
<ROW num="2">
    <ISBN>80-01-00704-9</ISBN>
    <TITLE>Programming Language C</TITLE>
    <PRICE>25</PRICE>
  </ROW>
  <!-other records -->
</ROWSET>
```

Fig. 3. XML data in Oracle format

Getting DTD or XML Schema according to the specified query could be acquired by calling specialized methods.

For updating and deleting data the application must specify key columns for searching corresponding records in the database. It also means that these statements can affect more than one row. For example, following code uses Oracle's database interface for inserting XML data in format shown in figure 3.

Update or delete processing follows the same code structure as in figure 4 using methods `updateXML(url)` and `deleteXML(url)` respectively. Before this calls the application has to set data key for looking up corresponding rows to process.

```
DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
Connection conn =
DriverManager.getConnection("jdbc:oracle:oci8:dp/tester@");

/* inserting into Book table */
OracleXMLSave sav = new OracleXMLSave(conn, "Book");

/* fileName contains a path to a XML data file */
URL url = sav.getURL(fileName);
rowCount = sav.insertXML(url);

System.out.println(rowCount + " were inserted.");
conn.close();
```

Fig. 4. Fragment of Java code inserting XML data into database

Sybase. Sybase ASE 12.5 is also shipped with some kind of XML support. In comparison with Oracle the library is small and allows only basic operations - it can just retrieve data based on SQL SELECT and insert data into database. No update or delete processing is possible.

Having the same query as in previous example, we get a XML document as follows in figure 5.

Unlike Oracle data format, this one contains metainformation about retrieved data. This metainformation is taken from the JDBC `ResultSetMetaData` class.

Both interfaces have an important drawback - data import can be done only to one table or view. Other cases must be processed by application logic in higher layer.

6 Use Case - Data Synchronization

Let us suppose a bit more difficult problem coming from a real life systems - an offline synchronization of data between two different versions of relational systems. This example uses Oracle 8i and Sybase ASE 12.5 as experimental systems.

For simplicity we can suppose that these systems have the same relational structure (same tables and their attributes). This requirement is however too strong as we will see later - but our simple XML-RDB columns mapping doesn't allow any differencies. With using an advanced mapping, names of attributes and tables can differ. Only the data types should correspond.

The algorithm for transferring data of one table (or view) works as follows:

1. Connection to both databases using a JDBC interface.
2. Getting XML data from both sources regarding SQL SELECT statements.
3. Transformation to common XML format for easier manipulation.

```
<ResultSet>
  <ResultSetMetaData getColumnCount="3">
    <ColumnMetaData getColumnDisplaySize="13" getColumnLabel="ISBN"
     getColumnName="ISBN" getColumnType="12" getPrecision="0"
     getScale="0" isAutoIncrement="false" isCurrency="false"
     isDefinitelyWritable="false" isNullable="false" isSigned="false"/>
    <!-  metadata for other columns ... -- >
  </ResultSetMetaData>
  <ResultSetData>
    <Row>
      <Column name="ISBN">80-85615-74-8</Column>
      <Column name="Title"> LaTeX for Beginners </Column>
      <Column name="Price">129.0</Column>
    </Row>
    <Row>
    <Column name="ISBN">80-01-00704-9</Column>
    <Column name="Title">Programming Language C</Column>
    <Column name="Price">25.00</Column>
    </Row>
  </ResultSetData>
</ResultSet>
```

Fig. 5. XML data in Sybase format

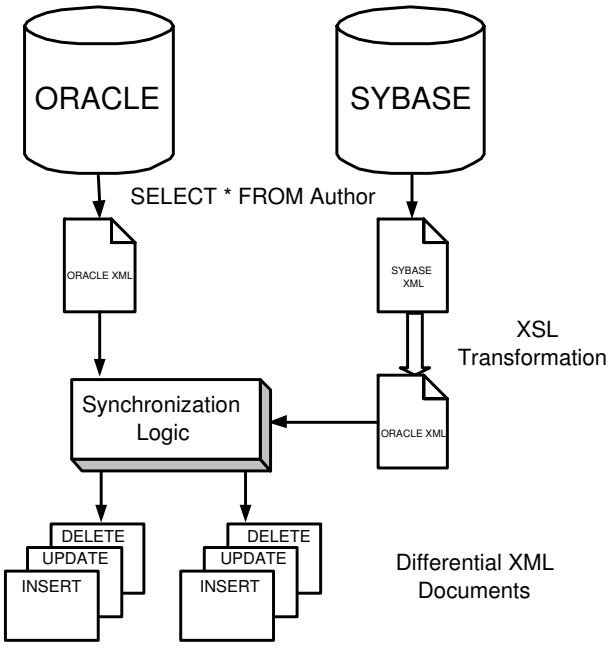


Fig. 6. Data synchronization diagram

4. Specification of key attributes for searching for corresponding records.
5. Searching for record sets for INSERT, UPDATE, DELETE statements on both sides (in figure 6 marked as Differential XML Documents). Looking up corresponding records is performed via XPath statements containing conditions with key values,
   e.g. `/ROWSET/ROW[ISBN="80-85615-74-8"]`.
6. Performing selected database operation using XML interfaces depending on task type.

This processing should be repeated sequentially for whole set of synchronized entities. Depending on relational schema we could encounter problems outlined in section 5.

Regarding of choosing common XML format (mentioned in step 3) there are many ways which structure of XML file to use. In this experiment the "native" Oracle XML format was chosen. The reason is simple. It is namely possible to perform a loss less transformation between Oracle and Sybase data formats. This transformation can also rename attributes what solves the problem with different database structures. Other renaming operations can be performed as aliases in SQL SELECTs.

Middleware Java libraries, XSL stylesheets and their detailed description can be downloaded at [1].

7 Conclusions

This paper describes basic issues of data transfer between relational databases using XML format. Concepts of native XML interfaces of most widely known commercial databases Oracle and Sybase have been also described.

As a significant part of this paper there is shown an example of data synchronization between two relational databases using XML. This example suggests a data transfer method using a XSLT transformation for loss less conversion between different output XML formats. Detailed description and sources can be downloaded at [1].

Future work in this area should be focused on exploration of sophisticated RDB - XML mapping. Current XML interfaces are not ready for non-trivial transfers yet.

References

1. Loupal, P.: *Metody prenosu dat prostrednictvim metadat.* Thesis. January, 2003. `http://cs.felk.cvut.cz/\char126\relaxloupalp/papers`
2. Bourret, R.: *XML-DBMS - Middleware for Transferring Data between XML Documents and Relational Databases.* `http://www.rpbourret.com/xmldbms`
3. World Wide Web Consortium. `http://www.w3.org`
4. World Wide Web Consortium Recommendation. *Extensible Markup Language (XML) 1.0* February, 1998. `http://www.w3.org/TR/REC-xml`

5. World Wide Web Consortium Recommendation. *Document Type Declaration.* February, 1998. http://www.w3c.org/TR/REC-xml#dt-doctype
6. World Wide Web Consortium Recommendation. *XSL Transformation (XSLT) Version 1.0.* November, 1999. http://www.w3.org/TR/xslt
7. World Wide Web Consortium Recommendation. *XML Path Language (XPath) Version 1.0.* November, 1999. http://www.w3.org/TR/xpath
8. World Wide Web Consortium. *XML Schema.* http://www.w3c.org/XML/Schema
9. Ball, S.: *XSLT Standard Library.* 2001, http://xsltsl.sourceforge.net
10. Oracle, *Oracle XML Developer's Kit for Java*, http://otn.oracle.com/tech/xml/xdk_java/content.html

Ontology Merging in Context of Web Analysis

Martin Labský and Vojtěch Svátek

Department of Information and Knowledge Engineering,
University of Economics, Prague, W. Churchill Sq. 4, 130 67 Praha 3, Czech Republic
{labsky|svatek}@vse.cz

Abstract. The *Rainbow* system aims at the analysis of websites by means of distributed modules specialized in particular types of data, such as free text, HTML structures or link topology. In order to ease the integration of services offered by the individual modules, which may come from third parties, a collection of ontologies has been developed. Parts of the ontologies contain information specific to the different ways of analyses, resulting in a need for integration. This paper describes how ontology–merging, namely the FCA–Merge method, may be used to integrate the results of multiple analyses for a certain application domain.

1 Introduction

The vast amount of knowledge present on the World Wide Web is still harder to track and process. The *Rainbow* system is a general framework aimimg at World Wide Web analysis from multiple data type specific viewpoints, we thus speak about *multiway* analysis. The proposed framework is an open architecture in which multiple modules performing different types of website analysis cooperate to achieve better results than if used separately. In this paper, we focus on the integration of results obtained from each data type specific analysis. Since the *Rainbow* system heavily exploits ontologies [8], our approach utilizes Formal Concept Analysis (FCA) which is closely related to ontology modeling. In particular we use an adapted FCA-Merge ontology merging method, described in [5].

From the user point of view, *Rainbow* offers its functionality to client applications via web services. In essence, three general types of services are offered to the clients - *classification of web documents*, *retrieval of web documents*, and *information extraction from web documents*. The intents of these services are in accord with their commonly used meanings, as described in [3]. So far we have focused on two use cases — pornography recognition, which is a special case of web document classification, and extraction of key facts from websites of organizations offering products and services (OOPS sites), a subtype of information extraction.

In section 2 we briefly describe the ontology–based architecture and current state of implementation of the *Rainbow* system. In section 3 we present our approach to integrating the data type specific results of multiple analyses. The last section briefly concludes and outlines the perspectives for the future.

2 Architecture of the *Rainbow* System

The central idea of *Rainbow* (Reusable Architecture for INtelligent Brokering Of Web information access) is firstly the *separation* of different ways of web analysis according to the syntactic *type of data* involved and, simultaneously, the *exploitation of mutual dependencies* between the results of data type specific analyses. In this way, the natural complementarity and/or supplementarity of information inferable from different types of data is used to provide extra information to clients as opposed to using single data type analyses.

2.1 Analysis Types

In the proposed architecture, all modules will utilize client–server model to communicate directly with clients but will also be able to communicate with each other, using each other's services as part of fulfilling their tasks. The functionality of *Rainbow* modules is offered by the means of *web services* defined by WSDL[1] interface descriptions. As a mediator we currently use the SOAP[2] protocol carried over HTTP on a strict request-response basis.

The currently planned suite of *Rainbow* components comprises modules for the analysis of six high–level data types: HTML structures, linguistic structures [2], explicit metadata (in META tags and possibly RDF [4]), URL addressing [6], link topology, and images. The current integrated prototype only includes simple forms of free text analysis, META tag and URL analysis; modules for link topology, HTML structure and image analysis already exist but have not been yet integrated into the architecture[3]. We suppose new modules will be added to *Rainbow* also by reusing existing systems. Since all modules must be able to communicate in a uniform way with *Rainbow* clients and other modules, a communication wrapper will typically be needed to make a third–party component available as a *Rainbow* module.

2.2 Conceptual Framework and Abstract Inferences

Ontology is typically defined as a shared formal conceptualization of a domain [8]. In *Rainbow*, ontologies provide the basis for the *communication of modules and clients*, enable *consistency checking of offered services* and provide support for the *integration of multiple types of analyses*, which is central to this text.

In order to present the integration of different analyses in detail, we first need to summarize the used conceptual framework of web information access, first introduced in [7]. In general, there are two kinds of concepts in *Rainbow* ontologies — syntactic *types* and semantic *classes*. Examples of types are *Document*,

[1] `http://www.w3.org/TR/2001/NOTE-wsdl-20010315`

[2] `http://www.w3.org/TR/SOAP/`

[3] The implementation also includes modules for source data acquisition (spidering, canonization to XHTML, provision via DBMS) as well as interaction with users (plug–in panel for a web browser) and external services.

DocumentFragment, *AbstractLink*, *XMLElement*, or *HTMLDocument*. Among classes, there are e.g. *HProductCatalogue*, *TLeafDocument*, *HProductDescription* or *LSentence*. In our view, classes differ from types by their correspondence to a certain type of analysis and by the lack of their exact and stable definitions. The above class names are prefixed with corresponding types of analyses - 'H' for HTML structure, 'T' for topology and 'L' for linguistic analysis. *Rainbow* ontologies model classes as subconcepts of types (e.g. *TLeafDocument* is a subconcept of *Document*). Currently, we distinguish between four types of classes according to their parent syntactic type — we recognize Document, DocumentFragment, DocumentCollection and AbstractLink classes.

Among the most widely used relations in *Rainbow* ontologies is the transitive *part–of* relation, e.g. *HProductDescription* may be *part–of* a *HProductCatalogue*. Concepts can be *adjacent* to each other, they may be *identified–by* some other concepts etc. Inverse relations are defined where possible.

2.3 System of Ontologies

The *Rainbow* ontology[4], which contains concepts, relations and service definitions mentioned in the previous section, cannot be monolithic. Instead, the distinction of *analysis types* as well as of *application domains* suggests natural decomposition into four layers of ontologies as depicted in Fig. 1. The upper two layers are domain–independent and therefore reusable by applications from all domains. The lower two layers add information specific to the domain of analysis, e.g. OOPS sites or web pornography.

Upper Web Ontology. The abstract *Upper Web Ontology* (UWO) provides a hierarchy of common Web–related concepts and relations that are shared by all analysis types and application domains. The UWO doesn't attempt to define an exhaustive description of the WWW. Instead, it provides a shared conceptual language for all *Rainbow* modules to build upon. The UWO defines only concepts that correspond to syntactic types as defined in section 2.2.

Partial Generic and Domain Web Models. For each way of analysis, *Partial Web Models* occupy the middle layers of the *Rainbow* ontology system. Concepts and relations defined in these models represent the *types* and *classes* specific to one particular way of analysis. The partial web models consist of a *generic* and *domain–dependent* part. Elements introduced in the generic model are based on the UWO and are reusable across different application domains. On the other hand, for each way of analysis there might be domain models specializing in different application domains. All of these domain models are then based on a single generic model and the common UWO. Concepts from the generic and domain models mostly correspond to *classes* of resources, but new *types* may be defined as well. In Fig. 1, the generic model and OOPS domain model for HTML

[4] The current version of *Rainbow ontologies* is available at `http://rainbow.vse.cz`.

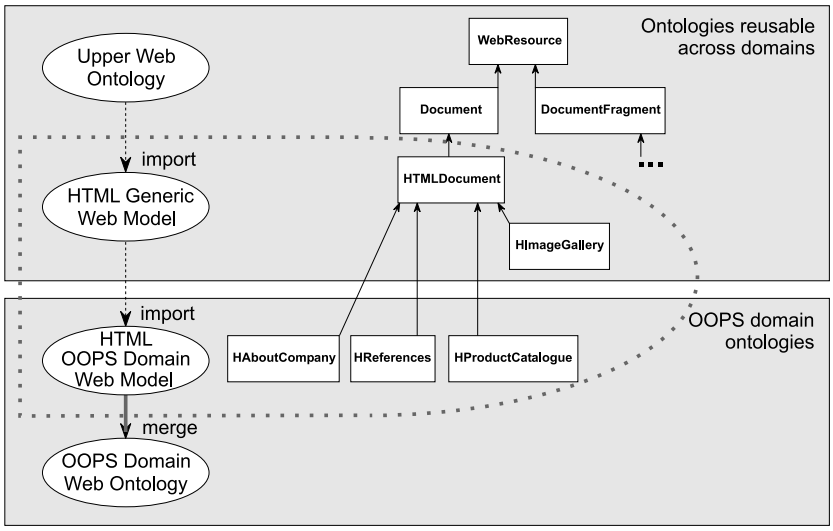


Fig. 1. Structure of the *Rainbow* ontology system shown on HTML analysis example

analysis are depicted within the dashed area. Examples of concepts from these models and the UWO are shown on the right.

As formal ontology language, we used *DAML+OIL*[5] (essentially, *RDFS*[6] as its sublanguage), and *OilED*[7] and *Protégé*[8] as ontology editors. The main advantage of *OilED* is the availability of a reasoner (FaCT), which enables to test the *consistency* of the model. On the other hand, *Protégé* offers a slightly more powerful interface capable of very simple ontology merging and visualization.

3 Integration of Multiple Analyses

As described in the previous section, the key information contained in partial web models (both generic and domain) are the concept hierarchies of semantic *classes*. Each such hierarchy extends one of the classified syntactic types, such as *Document* or *DocumentFragment*, and presents a data–type restricted view of that type. In our approach, the integration of multiple analyses consists in *merging* these class hierarchies[9]. The resulting class hierarchy is no more restricted to a single data–type view and is included in the Domain Web Ontology (DWO), as depicted in Fig. 1.

[5] `http://www.daml.org/2001/03/daml+oil-index.html`
[6] `http://www.w3.org/TR/rdf-schema/`
[7] `http://oiled.man.ac.uk`
[8] `http://protege.stanford.edu`
[9] These is–a hierarchies will often be flat, since the analysis–specific *classes* are usually not richly structured before integration.

The DWO also imports all concepts from the UWO and partial models that were not subject to merge. Thus, the DWO is a *domain-specific ontology of web resources*, and it *integrates all data–type specific views* of the used analyses. The integration of multiple types of analyses in context of a particular domain is motivated by the following benefits:

- First, *interesting relations* between analysis–specific classes may be revealed. These are most often subsumptions which may be characteristic just for the analyzed domain. Identical classes are merged.
- Second, the original data–type specific class hierarchies may be compared with the merged hierarchy to find *inconsistencies* in the original taxonomies.
- Third, the merged Domain Web Ontology could be used to provide *integrated access* to *Rainbow* services. In this case, *Rainbow* clients would not communicate with individual modules anymore. Instead, the merged DWO classes would be used for communication with the whole *Rainbow* system.

3.1 Deriving the Domain Web Ontology

We propose to derive the structure and content of the DWO based on labeled data, which may be acquired e.g. from training data previously used to train the individual modules. This labeled data comprises of training web pages that are simultaneously labeled by classes used by all different types of analyses.

Input Data to the Merge Process. As mentioned earlier, *Rainbow* currently addresses semantic classes that are subclassed from the following four syntactic types: *Document*, *DocumentFragment*, *DocumentCollection* and *AbstractLink*. Although the above four kinds of classes describe different objects, these four types of objects have certain ontology–defined relations (such as *part–of*). Using these relations, we can — in a limited way — relate the concepts to each other and use this information as input to the merge process.

Before we start merging concepts, we have to choose one *central* syntactic type to which all other (currently three) syntactic types will be related. For example, if the merge process is done for the Document as the central type, we regard the labeled input data as a set of training *Documents*. The input attributes used by the merge process are then of the following forms:[10]

- *is* <*Document* class>,
- *contains* <*DocumentFragment* class>,
- *part–of* <*DocumentCollection* class>, or
- (*source–of* or *target–of*) <*AbstractLink* class>.

For the other three syntactic types, the same relations (or their inverses) may be used to derive equivalent attribute sets. For example, if DocumentFragment

[10] *part–of*, *source–of*, *target–of* and their inverses are standard relations from the *Rainbow* ontology introduced by UWO.

was chosen as the central type, we would regard the labeled data as a set of *DocumentFragments* and use attributes such as *part–of* <*Document* class>).

In further text, we will only consider *Document* as the central syntactic type. The labeled input data is therefore treated as a set of documents, which we will denote as D. Each document $d_i \in D$ has attributes given by the analysis–specific classes it is labeled with. Each of these attributes has one of the above four forms.

The Merge Process. The task is essentially that of merging parts of multiple ontologies based on concept extensions. For its realization, we use an adapted FCA–Merge method introduced in [5], based on formal concept analysis (FCA). Due to limited space we cannot describe FCA in this text and we point the reader e.g. to [1]. FCA–Merge is a semi–automatic method which originally integrates ontologies based on a set of natural language documents, in which references to ontology concepts are found using NLU techniques. In our approach, we use a set of training documents D, where labeled instances of to–be–merged concepts already exist.

To perform integration, we need to obtain *formal contexts* for each type of analysis. A formal context is a triple (G, M, I), where G is a set of *objects*, M is a set of *attributes*, and I is a binary *incidence* relation such that $I \subseteq G \times M$.

Let A be the set of all analyses being integrated. For each analysis $a_i \in A$, we compute a formal context κ_i from the set of labeled documents D: $\kappa_i = (D, M_i, I_i)$, where D is a set of training documents, M_i is a set of classes specific to the i-th analysis, and $I_i \subseteq D \times M_i$ is the incidence relation, specifying which documents are labeled with which classes. The fact that $(d, m) \in I_i$ is read as "document d has the attribute m".

After computing formal contexts κ_i for each analysis a_i, these contexts are merged to produce a single formal context $\kappa = (D, M, I)$, where the set of labeled documents D remains unchanged, $M := \cup_{i=1}^{n}(M_i)$ where n is the number of different analyses, and $(d, m) \in I \Leftrightarrow (d, m) \in I_i$.

Pursuant to FCA–Merge, a *concept lattice*[11] is computed from the merged formal context κ. The concept lattice contains a system of *formal concepts*, which are used to construct the new ontology in a semi–automatic process according to FCA–Merge guidelines [5].

The result of the FCA–Merge process is a hierarchy of new concepts, each of which may be (1) an exact copy of its original analysis–specific concept, or (2) a concept that has been created by combining more of the original concepts together. FCA–Merge assists the human integrator and makes suggestions, but the ultimate modeling decisions are left to the integrator. Concepts can typically be combined when they have the same *extension* in the training data, resulting in them being replaced by a new concept in the role of their conjunction. In case two or more of the original concepts have a significant overlap, a new concept, representing their conjunctions, may be added to the new ontology as well as the original ones. This takes place especially when there exist concepts in the source ontologies that are more specific than the potential new concept, and/or when

[11] For definitions of FCA–related terminology, see [5] or [1].

the new concept has a high support[12] in the training data. The hierarchy of the new ontology is derived directly from the concept lattice and can be altered by the integrator. Typically, the resulting class hierarchy is deeper (as it is drawn from data) than the usually flat hierarchy of input classes.

The new concept hierarchy reveals previously hidden subsumption relations across the different analysis types. These empirically induced subsumptions may further be checked with formal ontology definitions of the original concepts. This checking could be made automatic and a conflict would signal either inconsistently labeled training data or a wrong definition of some of the original concepts.

Another output of the merge process are two mapping functions between the set of the original concepts M and the set of merged concepts, which we will denote as C. The first function we call $targets : M \Rightarrow P(C)$, as it transforms an original concept into a set of concepts that were created from it. The other function, $sources : C \Rightarrow P(M)$, transforms a merged concept into a set of concepts it was created from.

The final DWO consists of the merged *classes*, while all information other than the original *classes* is simply imported from the partial web models and the UWO. In principle, the DWO can be used for two purposes. First, we may just want to examine the domain, find hidden relationships in data and use this information for e.g. tuning modules' knowledge bases, adapting original concept definitions, correcting training data or adapting applications.

Second, the UWO may be used for a completely different purpose — as an integration and communication platform for applications that want to use the *Rainbow* system as a whole. Instead of calling the services of individual modules and working with their analysis–specific concepts, applications may prefer to work with the merged concept hierarchy, as it better reflects the relations in the training data[13] (and thus in the domain, presumably).

The merged DWO concepts cannot be used to query the modules directly, since modules don't know the merged concept language. Instead, there must be a mechanism based on the abovementioned functions *targets* and *sources* and their inverses, which will translate the DWO concepts to the original concepts understood by the modules and back. This task will be taken care of by a dedicated integration module, which will manage the bidirectional translation of concepts and control the execution of the correct modules[14] . This part of our research is currently in an early phase of development.

---

[12] We compute support as the number of positive documents divided by the total number of training documents.

[13] Large amount of training data with enough information about all analysis types will be required for this task.

[14] A single application request will typically need to be translated into a control structure containing multiple requests upon the *Rainbow* modules. The result will then transparently (to the application) integrate multiple analysis views.

3.2 Example

In the following we present a 'toy' example taken from the domain of websites of car dealers. In the example, integration is done based on a set of only thirteen manually labeled documents. Nevertheless, we will be able to demonstrate some of the benefits of this method. Due to lack of space, we will consider only one kind of semantic classes — Document classes. The other three kinds of classes could be taken into account as described in section 3.1. We also limit the integration to just two types of analyses — HTML structures and topology.

Table 1. HTML documents classified by the topology and HTML structure analysis

I	TLE	THU	TLO	TRE	HAB	HRE	HPR	HIM
d1	×				×			
d2		×	×		×			
d3	×				×			
d4		×		×		×		
d5		×		×		×		
d6		×	×				×	×
d7		×		×			×	×
d8		×	×				×	×
d9		×	×				×	×
d10		×		×			×	
d11		×	×				×	
d12		×		×			×	
d13							×	

The above table shows thirteen documents classified by the following data type specific classes. Topology–specific classes are prefixed with 'T', while HTML–specific classes have 'H' as their first letter[15] :

- *TLeaf* (TLE) is a document with no links to other documents,
- *THub* (THU) has more than 1 link to other documents,
- *TLocalHub* (TLH) is a subconcept of THub, most of whose links lead to the same IP address as its own,
- *TRemoteHub* (TRH) is a subconcept of THub, most of whose links lead to a different IP address than its own,
- *HAboutCompany* (HAB) is a document describing a company in general,
- *HProductCatalogue* (HPR) contains descriptions of offered products,
- *HReferences* (HRE) refers to customers' pages,
- *HImageGallery* (HIM) contains a set of similarly–sized images.

From the labeled set of documents, we are able to construct a concept lattice depicted in Fig. 2. Formal concepts that are on the same level of specificity as the source class hierarchies are shown in dark color. On the other hand, formal concepts more specific than original classes are drawn in light color.

[15] The presented concept intents are only informative, exact meaning is specified by modules performing the analysis.

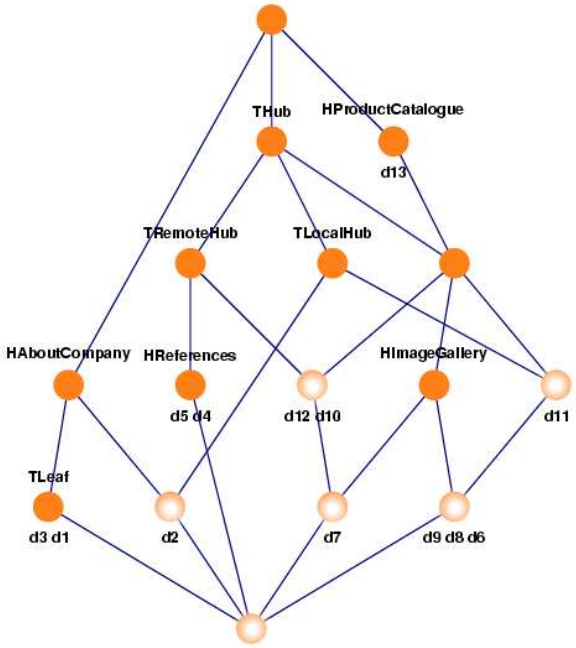


Fig. 2. Pruned concept lattice derived for the 13 car dealer documents

We observe that no original concepts had the same extensions in the training data, as each formal concept directly corresponds to at most one ontology concept. Depending on the use of the target ontology, all or most of the original classes could be included in it. In the concept lattice, we might reveal several interesting implications pertaining the analyzed domain (or, more precisely, the training data), such as $HImageGallery \Rightarrow HProductCatalogue$ or $HReferences \Rightarrow HRemoteHub$. It is up to the integrator what part of the induced hierarchy will be reflected in the target ontology.

We can also identify one new (no-name) concept which is a combination of *THub* and *HProductCatalogue* and whose specialization is *HImageGallery*. This concept might e.g. stand for product catalogues referring to child documents with detailed information about the sold products — let us denote it *ComplexProductCatalogue*[16] . Because of its interpretation and being within the specificity level of the original ontologies, *ComplexProductCatalogue* is an appropriate candidate for adding to the target ontology.

Last of all, formal concepts more specific than the original ontologies should be examined. Frequently occurring combinations of original ontology concepts may be included in the target ontology as well if they are considered useful by human integrators. For example, the formal concept with extension $d6, d8, d9$ and

[16] The new concept is not specific to any type of analysis.

significant support of 0.23 may be considered interesting for certain applications and included in the target ontology as *GraphicalLocalCatalogue*.

4 Conclusions and Future Work

In this text, we demonstrated the use of ontology merging method FCA–Merge [5] for the integration of multiple ways of web analyses. We provided a simplified example of merging class hierarchies used by HTML structure analysis and topology analysis and created a new merged ontology based on a collection of labeled web pages. We also briefly presented the system of *Rainbow* ontologies that were subject to the merge process.

The most imminent direction for future work is testing the presented approach on a large collection of training data, which will first be taken from small segments of the OOPS domain. In conjunction with testing, we will check the feasibility of building an integrated interface to *Rainbow* services using the presented method. Apart from the described FCA–based approach to knowledge integration, we also plan to exploit the valuable information stored in probability distributions among different analysis–specific classes in the training data. The evolving prototype implementation of *Rainbow* should constantly serve for empirical validation of (not only) the mentioned research directions.

Acknowledgements

The authors would like to thank their collaborators in the *Rainbow* project, who contributed to the development of ontologies—Miroslav Vacura, Jirka Kosek, Martin Kavalec, Martin Sajal and Petr Berka. The research has been supported by grant no. 201/00/D045 of the Grant Agency of Czech Republic.

References

1. Beneš, M., Snášel, V.: Deducing Design Class Hierarchy from Object Properties. In: (Hanáček, P., ed.) Information Systems Modeling, Ostrava 2002, 203–211.
2. Kavalec, M., Svátek, V.: Information Extraction and Ontology Learning Guided by Web Directory. In: (Aussenac-Gilles, N., Maedche, A., eds.) ECAI Workshop on NLP and ML for Ontology Engineering. Lyon, 2002, 39–42.
3. Kosala, R., Blockeel, H.: Web Mining Research: A Survey. In: ACM SIGKDD Explorations, 2(1):1-15, 2000.
4. Lassila, O., Swick, R.: Resource Description Framework (RDF) Model and Syntax Specification. Recommendation, World-Wide Web Consortium, Feb. 1999.
5. Stumme, G., Maedche, A.: FCA–Merge: A Bottom–Up Approach for Merging Ontologies. In: IJCAI '01 - Proceedings of the 17th International Joint Conference on Artificial Intelligence, Morgan Kaufmann 2001.
6. Svátek, V., Berka, P.: URL as starting point for WWW document categorisation. In: (Mariani, J., Harman, D.:) RIAO'2000 – Content-Based Multimedia Information Access, CID, Paris, 2000, pp.1693–1702.

7. Svátek, V., Kosek, J., Bráza, J., Kavalec, M., Klemperer, J., Berka, P.: Framework and Tools for Multiway Extraction of Web Metadata. In: (Hanáček, P., ed.) Information Systems Modelling, Ostrava 2002, 235–242.
8. Uschold, M., Jasper, R.: A Framework for Understanding and Classifying Ontology Applications. In: IJCAI'99 Workshop on Ontologies and Problem-Solving Methods: Lessons Learned and Future Trends. Stockholm 1999.

Web Services and WSDL*

Karel Richta

Dept. of Computer Science, Engineering Faculty of Electrical Engineering
Czech Technical University of Prague
Karlovo nám. 13, Praha 2, Czech Republic
richta@fel.cvut.cz

Abstract. The paper deals briefly with WSDL (Web Services Definition Language) as a tool for the specification of web services. The main purpose for WSDL is something similar to XML for the transfer of data. The example of the small web service and its call is used to explain how WSDL works.

Key words: WSDL,XML,HTTP,UDDI

1 Introduction

In the present time, it is hard to imagine computing without the Web. The reason why the Web succeeded seems to be simplicity and global accessibility. From a service provider's point of view (e.g. an e-shop provider), they can join the global community. From a client's point of view (e.g. an e-shop buyer), you can access services through Web. The majority of the web's work is done by HTTP methods `GET`, `POST`, and `PUT`, and a simple markup language XML. The web services are the result of the fact that the advantages of the Web as a platform apply not only to documents but also to services. A more formal definition of a web service may be borrowed from [11] "Web services are a new breed of Web application. They are self-contained, self-describing, modular applications that can be published, located, and invoked across the Web. Web services perform functions, which can be anything from simple requests to complicated business processes...Once a Web service is deployed, other applications (and other Web services) can discover and invoke the deployed service." There exists different middleware platforms: CORBA, DCOM, RMI, Jini, etc. While middleware platforms provide great implementation vehicles for services, none of them is a clear winner. The strengths of the Web is its "simplicity of access and ubiquity". The Web provides a uniform and widely accessible interface and access, although middleware services are more efficiently implemented in a traditional middleware platform.

* This work has been partially supported by the research program no. MSM 212300014 "Research in the Area of Information Technologies and Communications" of the Czech Technical University in Prague (sponsored by the Ministry of Education, Youth and Sports of The Czech Republic), and it also has been partially supported by the grant No. 201/03/0912 "XML Documents Searching and Indexing" of the Grant Agency of The Czech Republic.

2 What Web Services Are Based On?

The basic platform for Web services is XML and HTTP. HTTP is a protocol, running practically everywhere on the Internet. XML provides a meta-language in which you can write specialized languages (called XML applications) to express complex interactions between clients and services or between components of a composite service. Behind the front wall of a web server, the XML message gets converted to a middleware request and the results converted back to XML. The resulting infrastructure is similar to CORBA, i.e. IDL plus remote procedure calls. But the real life is never quite that simple, and there are many problems with the platform supported. On the other hand, the Web needs to be augmented with a few other platform services, which maintain the simplicity of the Web, to constitute a more functional platform. The Web services are based on XML - everything is converted into XML (data, function calls, etc.). XML data are transferred with the help of HTTP protocol. Function calls are expressed in an XML application called SOAP (Simple Object Access Protocol [9]). The SOAP document describes, what service, and with which parameters is required. The characteristics of the service should be described somehow. That is duty of WSDL (Web Services Description Language [13]), the other XML application. The complete list of available services is something like directory service, and it is called UDDI (Universal Description, Discovery and Integration Service [10]. At higher levels, one might also add technologies such as XAML, XLANG (transactional support for complex web transactions involving multiple web services), XKMS, and XFS (XML Key Management Specification) - services that are not universally accepted as mandatory.

3 Simple Object Access Protocol SOAP

SOAP [9] is a simple protocol specification that defines a uniform way of passing XML-encoded data. It also defines a way to perform remote procedure calls (RPCs) using HTTP as the underlying communication protocol. SOAP arises from the understanding that no matter how clever the current middleware offerings are, they need a WAN wrapper. Architecturally, sending messages as plain XML has advantages in terms of ensuring interoperability. The top element of the SOAP XML document representing the message is the Envelope element. The Envelope element may contain the Header element, must contain the Body element, and may contain the Fault element. The SOAP Fault element is used to carry error and/or status information within a SOAP message. If present, the SOAP Fault element must appear as a body entry and must not appear more than once within a Body element. The SOAP message containing request for a local time can look like follows:

```
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
    <SOAP-ENV:Body>
```

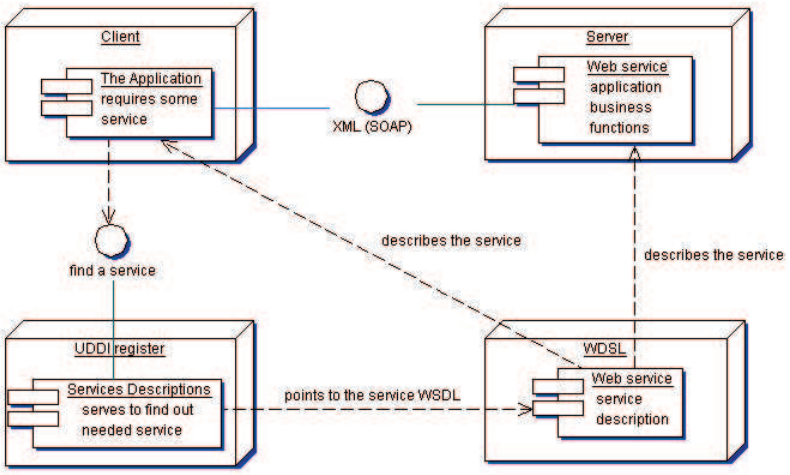


Fig. 1. The SOAP document structure

```
        <time:GetLocalTime xmlns:time="www.zvon.org/time">
             <time:city>New York</time:city>
             <time:format template="hh:mm"/>
        </time:GetLocalTime>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

4 WSDL: Web Services Definition Language

WSDL [13] provides a way for service providers to describe the basic format of web service requests over different protocols or encodings. WSDL is used to describe *what* a web service can do, *where* it resides, and *how* to invoke it. While the claim of SOAP/HTTP independence is made in various specifications, WSDL makes the most sense if it assumes SOAP/HTTP/MIME as the remote object invocation mechanism. UDDI registries describe numerous aspects of web services, including the binding details of the service. WSDL fits into the subset of a UDDI service description.

WSDL defines services as collections of network endpoints or *ports*. In WSDL the abstract definition of endpoints and messages is separated from their concrete network deployment or data format bindings. This allows the reuse of abstract definitions of messages, which are abstract descriptions of the data being exchanged, and port types, which are abstract collections of operations. The concrete protocol and data format specifications for a particular port type constitute a reusable binding. A port is defined by associating a network address

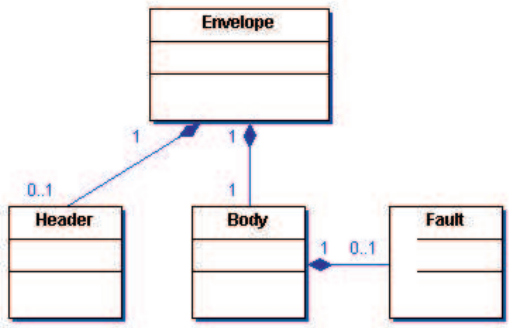


Fig. 2. The WSDL document structure

with a reusable binding; a collection of ports define a service. And, thus, a WSDL document uses the following elements in the definition of network services:

- Types – a container for data type definitions using some type system (such as XSD).
- Message – an abstract, typed definition of the data being communicated.
- Operation – an abstract description of an action supported by the service.
- Port Type – an abstract set of operations supported by one or more endpoints.
- Binding – a concrete protocol and data format specification for a particular port type.
- Port – a single endpoint defined as a combination of a binding and a network address.
- Service – a collection of related endpoints.

So, in plain English, WSDL is a template for how services should be described and bound by clients. In what follows, I've described a stock quote service advertisement and a sample request/response pair for the service, which seeks the current quote on Motorola (ticker: MOT).

4.1 The WSDL example

Let us suppose we want to create a simple Web service to compute sum of two integers [2]. In the Message part it has to be declared, the sum will be requested with two parameters p0 and p1 of the integer type:

```
<wsdl:message name='Sum_Request'>
  <wsdl:part name='p0' type='xsd:int'/>
  <wsdl:part name='p1' type='xsd:int'/>
</wsdl:message>
```

The response will again be an integer:

```
<wsdl:message name='Sum_Response'>
  <wsdl:part name='response' type='xsd:int'/>
</wsdl:message>
```

In the portType part it has to be set, that the operation Sum with parameters p0 and p1 in this order will be called:

```
<wsdl:portType name='Sum'>
  <wsdl:operation name='Sum' parameterOrder='p0 p1'>
    <wsdl:input name='Sum' message='tns:Sum_Request'/>
    <wsdl:output name='Sum' message='tns:Sum_Response'/>
  </wsdl:operation>
</wsdl:portType>
```

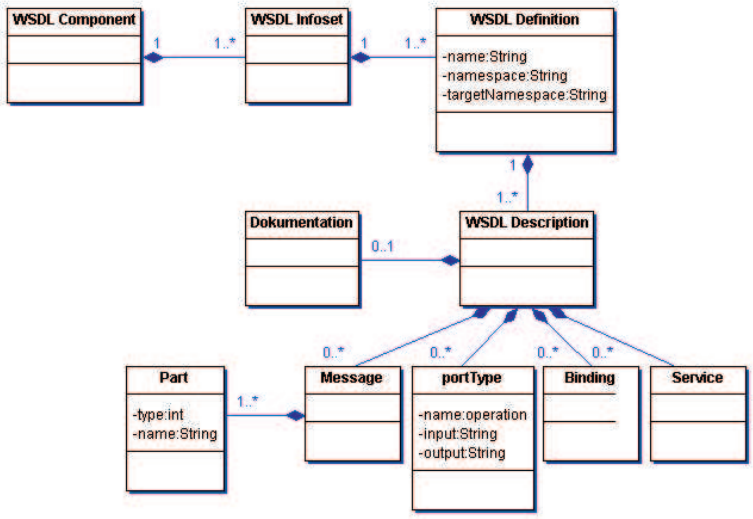


Fig. 3. WSDL and UDDI deployment

In the Binding part it has to be set, that the operation call will be transported via RPC encoded according to appropriate XML schema as an input and output elements:

```
<wsdl:binding name='SumSOAPBinding0' type='tns:Sum'>
  <soap:binding transport='http://schemas.xmlsoap.org/soap/http'
    style='rpc'/>
  <wsdl:operation name='Sum'>
```

```
    <soap:operation soapAction='' style='rpc'/>
    <wsdl:input name='Sum'>
      <soap:body use='encoded'
        encodingStyle='http://schemas.xmlsoap.org/soap/encoding/'
        namespace='urn:x-kosek:services:Sum'/>
    </wsdl:input>
    <wsdl:output name='Sum'>
      <soap:body use='encoded'
        encodingStyle='http://schemas.xmlsoap.org/soap/encoding/'
        namespace='urn:x-kosek:services:Sum'/>
    </wsdl:output>
  </wsdl:operation>
</wsdl:binding>
```

In the Service part it has to be set, which node will be addressed:

```
<wsdl:service name='Sum'>
  <wsdl:port name='Sum' binding='tns:SumSOAPBinding0'>
    <soap:address location='http://localhost:6060/Sum/'/>
  </wsdl:port>
</wsdl:service>
```

The whole picture look as follows:

```
<?xml version='1.0'?>
<wsdl:definitions name='Sum'
  targetNamespace='urn:x-kosek:services:Sum'
  xmlns:wsdl='http://schemas.xmlsoap.org/wsdl/'
  xmlns:xsd='http://www.w3.org/2001/XMLSchema'
  xmlns:tns='urn:x-kosek:services:Sum'
  xmlns:http='http://schemas.xmlsoap.org/wsdl/http/'
  xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
  xmlns:mime='http://schemas.xmlsoap.org/wsdl/mime/'
  xmlns:soap='http://schemas.xmlsoap.org/wsdl/soap/'
  xmlns:SOAP-ENC='http://schemas.xmlsoap.org/soap/encoding/'>
  <wsdl:message name='Sum_Response'>
    <wsdl:part name='response' type='xsd:int'/>
  </wsdl:message>
  <wsdl:message name='Sum_Request'>
    <wsdl:part name='p0' type='xsd:int'/>
    <wsdl:part name='p1' type='xsd:int'/>
  </wsdl:message>
  <wsdl:portType name='Sum'>
    <wsdl:operation name='Sum' parameterOrder='p0 p1'>
      <wsdl:input name='Sum' message='tns:Sum_Request'/>
      <wsdl:output name='Sum' message='tns:Sum_Response'/>
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:binding name='SumSOAPBinding0' type='tns:Sum'>
    <soap:binding
```

```
     transport='http://schemas.xmlsoap.org/soap/http' style='rpc'/>
    <wsdl:operation name='Sum'>
      <soap:operation soapAction='' style='rpc'/>
      <wsdl:input name='Sum'>
        <soap:body use='encoded'
          encodingStyle='http://schemas.xmlsoap.org/soap/encoding/'
          namespace='urn:x-kosek:services:Sum'/>
      </wsdl:input>
      <wsdl:output name='Sum'>
        <soap:body use='encoded'
          encodingStyle='http://schemas.xmlsoap.org/soap/encoding/'
          namespace='urn:x-kosek:services:Sum'/>
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:service name='Soucet'>
    <wsdl:port name='Soucet' binding='tns:SumSOAPBinding0'>
      <soap:address location='http://localhost:6060/Sum/'/>
    </wsdl:port>
  </wsdl:service>
</wsdl:definitions>
```

4.2 Web Services in the Action

At the server side the service will be implemented e.g. as:

```
public class Sum
{
  public int Sum(int x, int y) { return x + y; }
}
At the client side the service will be declared:
package client.iface;
public interface Sum
{
    int Sum(int p0, int p1);
}
```

The client application that uses this service can look as follows:

```
package client;

import client.iface.*;
import org.idoox.wasp.Context;
import org.idoox.wasp.MessageAttachment;
import org.idoox.webservice.client.WebService;
import org.idoox.webservice.client.WebServiceLookup;

public class SumClient{
  public static void main(String args[]) throws Exception {
    String host = "http://localhost:6060/Sum/";

    //init the lookup
    WebServiceLookup lookup = (WebServiceLookup)Context.getInstance
        ("org.idoox.webservice.client.WebServiceLookup");

    //get the instance of the Web Service interface from the lookup
    //change the interface class to your Web Service's interface
    Sum service = (Sum)lookup.lookup
        ("http://localhost:6060/Sum/", Sum.class,host);

    //now call the methods on your Web  Service's interface
    System.out.println(service.Sum(2,5));
  }
}
```

The request will be enveloped into the SOAP envelope (where references to XML Schema types should be included):

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Body
    SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:wn1="http://www.w3.org/2000/10/XMLSchemas"
    xmlns:wn0="http://www.w3.org/2001/XMLSchemas"
    xmlns:xsi="http://www.w3.org/2001/XMLSchemas-instance">
      <nsp:Sum xmlns:nsp="urn:x-kosek:services:Sum">
        <p0 xsi:type="wn0:int">2</p0> <p1 xsi:type="wn0:int">5</p1>
      </nsp:Sum>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

And send to the server. When the sum is computed, the result is again enveloped into the SOAP envelope and returned back:

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Body
    SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:wn1="http://www.w3.org/2000/10/XMLSchemas"
    xmlns:wn0="http://www.w3.org/2001/XMLSchemas"
    xmlns:xsi="http://www.w3.org/2001/XMLSchemas-instance">
      <nsp:SumResponse xmlns:nsp="urn:x-kosek:services:Sum">
        <response xsi:type="wn0:int">7</response>
      </nsp:SumResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

4.3 Another Example: Service Advertisement

```
<?xml version="1.0"?>
<definitions name="StockQuote"
               targetNamespace="http://example.com/stockquote.wsdl"
               xmlns:tns="http://example.com/stockquote.wsdl"
               xmlns:xsd1="http://example.com/stockquote.xsd"
               xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
               xmlns="http://schemas.xmlsoap.org/wsdl/">
   <types>
    <schema targetNamespace="http://example.com/stockquote.xsd"
            xmlns="http://www.w3.org/1999/XMLSchema">
     <element name="TradePriceRequest">
      <complexType>
       <all>
        <element name="tickerSymbol" type="string"/>
       </all>
      </complexType>
     </element>
     <element name="TradePrice">
      <complexType>
       <all>
        <element name="price" type="float"/>
       </all>
      </complexType>
     </element>
    </schema>
   </types>

  <message name="GetLastTradePriceInput">
    <part name="body" element="xsd1:TradePriceRequest"/>
  </message>
  <message name="GetLastTradePriceOutput">
    <part name="body" element="xsd1:TradePrice"/>
  </message>
```

```
 <portType name="StockQuotePortType">
  <operation name="GetLastTradePrice">
   <input message="tns:GetLastTradePriceInput"/>
    <output message="tns:GetLastTradePriceOutput"/>
  </operation>
 </portType>

 <binding name="StockQuoteSoapBinding"
         type="tns:StockQuotePortType">
 <soap:binding style="document"
  transport="http://schemas.xmlsoap.org/soap/http"/>
   <operation name="GetLastTradePrice">
    <soap:operation
     soapAction="http://example.com/GetLastTradePrice"/>
      <input>
        <soap:body use="literal"
         namespace="http://example.com/stockquote.xsd"
         encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
      </input>
     <output>
      <soap:body use="literal"
        namespace="http://example.com/stockquote.xsd"
        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
     </output>
    </operation>
 </binding>

 <service name="StockQuoteService">
  <documentation>My first service</documentation>
   <port name="StockQuotePort" binding="tns:StockQuoteBinding">
    <soap:address location="http://example.com/stockquote"/>
   </port>
 </service>
</definitions>

 <binding name="StockQuoteServiceBinding"
  type="StockQuoteServiceType">
   <soap:binding style="rpc"
    transport="http://schemas.xmlsoap.org/soap/http"/>
<operation name="getQuote">
 <soap:operation
      soapAction="http://www.getquote.com/GetQuote"/>
       <input>
        <soap:body type="InMessageRequest"
         namespace="urn:live-stock-quotes"
         encoding="http://schemas.xmlsoap.org/soap/encoding/"/>
       </input>
       <output>
        <soap:body type="OutMessageResponse"
         encoding="http://schemas.xmlsoap.org/soap/encoding/"/>
```

```
        </output>
      </operation>
    </binding>
    <service name="StockQuoteService">
     <documentation>My first service
 </documentation>
      <port name="StockQuotePort"
       binding="tns:StockQuoteBinding">
       <soap:address location="http://example.com/stockquote"/>
      </port>
     </service>
</definitions>
```

A SOAP enveloped request to the StockQuote service

```
POST /StockQuote HTTP/1.1
Host: www.stockquoteserver.com
Content-Type: text/xml;
charset="utf-8"
Content-Length: nnnn
SOAPAction: "Some-URI"

<SOAP-ENV:Envelope
 xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
 SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
   <SOAP-ENV:Body>
    <m:GetLastTradePrice xmlns:m="Some-URI">
      <symbol>MOT</symbol>
    </m:GetLastTradePrice>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

A SOAP enveloped response to the StockQuote service

```
HTTP/1.1 200 OK Content-Type: text/xml; charset="utf-8"
Content-Length: nnnn

<SOAP-ENV:Envelope
 xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
 SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
 <SOAP-ENV:Body>
  <m:GetLastTradePriceResponse xmlns:m="Some-URI">
   <Price>14.5</Price>
  </m:GetLastTradePriceResponse>
 </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

5 UDDI, The Universal Description, Discovery and Integration Service

UDDI [10] provides a mechanism for clients to dynamically find other web services. Using a UDDI interface, businesses can dynamically connect to services provided by external business partners. A UDDI registry is similar to a CORBA trader, or it can be thought of as a DNS service for business applications. A UDDI registry has two kinds of clients: businesses that want to publish a service (and its usage interfaces), and clients who want to obtain services of a certain kind and bind programmatically to them. The UDDI contains so called "White pages", which contains information such as the name, address, telephone number, and other contact information of a given business. It tells how the provider of a Web service registers itself. The second part are "Yellow pages", which is a specification, how an application finds a particular Web service. The third part are "Green pages", where are the technical details necessary to invoke a Web service. This includes URLs, information about method names, argument types, and so on.

UDDI Example

Query: The following query, when placed inside the body of the SOAP envelope, returns details on Microsoft.

```
<find_business generic="1.0" xmlns="urn:uddi-org:api">
  <name>Microsoft</name>
</find_business>
```

Result: detailed listing of ¡businessInfo¿ elements currently registered for Microsoft, which includes information about the UDDI service itself.

```
<businessList generic="1.0"
  operator="Microsoft Corporation"
  truncated="false"
  xmlns="urn:uddi-org:api">
  <businessInfos>
   <businessInfo
       businessKey="0076B468-EB27-42E5-AC09-9955CFF462A3">
    <name>Microsoft Corporation</name>
     <description xml:lang="en">
            Empowering people through great software -
            any time, any place and on any device is Microsoft's
            vision. As the worldwide leader in software for personal
            and business computing, we strive to produce innovative
            products and services that meet our customer's
     </description>
      <serviceInfos>
       <serviceInfo
          businessKey="0076B468-EB27-42E5-AC09-9955CFF462A3"
```

```
          serviceKey="1FFE1F71-2AF3-45FB-B788-09AF7FF151A4">
        <name>Web services for smart searching</name>
       </serviceInfo>
      <serviceInfo
          businessKey="0076B468-EB27-42E5-AC09-9955CFF462A3"
          serviceKey="8BF2F51F-8ED4-43FE-B665-38D8205D1333">
        <name>Electronic Business Integration Services</name>
      </serviceInfo>
      <serviceInfo
          businessKey="0076B468-EB27-42E5-AC09-9955CFF462A3"
          serviceKey="611C5867-384E-4FFD-B49C-28F93A7B4F9B">
    <name>Volume Licensing Select Program</name>
      </serviceInfo>
  <serviceInfo
          businessKey="0076B468-EB27-42E5-AC09-9955CFF462A3"
          serviceKey="A8E4999A-21A3-47FA-802E-EE50A88B266F">
    <name>UDDI Web Sites</name>
      </serviceInfo>
      </serviceInfos>
    </businessInfo>
  </businessInfos>
 </businessList>
```

6 Conclusion

The paper try to show how WSDL serves for the description of service call. It means, that WSDL covers the syntax only. What remains to solve is the description of the semantics of web services. In the presented example we have to know, that the right function is the function "Sum". To be able to find out any function that sums two integer numbers is the task for the future.

References

1. V. Bisová, K. Richta: Transformation of UML Models into XML. In *Proceedings of Challenges 2000 ADBIS-DASFAA*. Praha: MATFYZPRESS UK, ISBN 80-85863-56-1, pp. 33-45. Praha 2000.
2. J. Kosek: *Inteligentní podpora navigace na WWW s využitím XML*. Diplomová práce KIT VŠE Praha, 2002, http://www.kosek.cz/diplomka/, (in Czech).
3. M. Nič: *SOAP Reference*. http://www.zvon.org/xxl/soapReference/
4. J. Pokorný, K. Richta: XML a semistrukturovaná data. In *Proceedings of DATASEM 2000*. Brno: Masaryk University - ISBN 80-210-2428-3, pp. 47–63. Brno 2000.
5. K. Richta: Using XSL in IS Development. In: *New Perspectives on Information Systems Development: Theory, Methods, and Practice*. New York : Kluwer Academic/Plenum Publishers, 2002, pp. 309-319. ISBN 0-306-47251-1.
6. K. Richta: Types in XML and XML-schemas. In *DATESO'01 – Proceedings of Workshop on Databases, Texts, Specifications, and Objects*. Prague : CTU, 2001, pp. 20-32. ISBN 80-01-02376-1. (in Czech).

7. K. Richta, P. Long: Using XSLT for IS Simulation. In *DATESO'02 – Proceedings of Workshop on Databases, Texts, Specifications, and Objects*. Ostrava : VŠB-TUO, 2002, pp. 66-78. ISBN 80-248-0080-2.
8. K. Richta, M. Badawy: Deriving Triggers from UML/OCL Specification. In *ISD 2002*. New York : Kluwer Academic/Plenum Publishers, 2002, pp.305–316. ISBN 0-306-47698-3.
9. *SOAP Version 1.2 Part 0: Primer*, W3C Candidate Recommendation 19 December 2002. `http://www.w3.org/TR/soap12-part0/`
10. *OASIS Committee Specifications, Universal Description, Discovery and Integration, UDDI Version 3.0*, Published Specification, 19 July 2002, `http://uddi.org/specification.html`
11. V. Vasudevan: *A Web Services Primer*. `http://webservices.xml.com/pub/a/ws/2001/04/04/webservices/`
12. W3C: *Web Services Activity*. `http://www.w3.org/2002/ws/`
13. *Web Services Description Language (WSDL) Version 1.2*, W3C Working Draft 3 March 2003, `http://www.w3.org/TR/wsdl12/`

Metadata Driven Data Pre-processing for Data Mining

Petr Aubrecht, Petr Mikšovský, and Zdeněk Kouba

Czech Technical University, Prague, Czech Republic,
{aubrech,miksovsp,kouba}@labe.felk.cvut.cz

Abstract. It is well known that success of every data mining algorithm is strongly dependent on a quality of processed data. In this context it is natural that data pre-processing can be a very complicated task. Sometimes, data pre-processing takes more than half of the total time spent by solving the data mining problem. The paper describes a tool called SumatraTT, the goal of which is to make the process of data pre-processing easier and faster.
Basically, SumatraTT is a metadata-driven, platform independent, extensible, and universal data processing tool. These features have been achieved by building the tool as an interpreter of a transformation-oriented scripting language called SumatraScript. SumatraScript is fully interpreted language with Java-like syntax combining together data access, metadata access, and common programming constructions. Furthermore, it supports RAD (Rapid Application Development) technology by providing the library of re-usable transformation templates.
The second part of the paper contains a practical application of SumatraTT. It is a task aimed at prediction of water consumption in a regional distribution network.

1 Introduction

Real-life data rarely complies with the requirements of various data mining tools. It is often inconsistent, noisy, they contain redundant attributes, they have unsuitable format, etc. That is why it has to be prepared carefully before the process of data mining can be started. It is well known that data preparation is a key to the success of data mining tasks. The "no free lunch theorem" actually tells that a clever pre-processing can compensate a nonoptimal choice of a mining algorithm for the given task. To support this argument Pyle [5] estimates percentage of time necessary to complete a data-mining project for various activities as well as importance of these activities (Tab. 1). Nevertheless, in past most effort was devoted to the development of more and more sophisticated mining tools while data pre-processing issues were considered as peripheral.

A universal data pre-processing tool should be able to help to a data miner with many every-day problems. For instance, data is provided in many different formats (text files, database files, etc.), different formats of values (national formats of date, time, currencies, etc), calculation of derived attributes, data filtering, joining several data sets, etc. The data mining process usually starts with

© J. Pokorný, V. Snášel (Eds.): Dateso 2003, pp. 55–62, ISBN 80-248-0330-5.
VŠB – Technical University of Ostrava, Dept. of Computer Science, 2003.

Table 1. Stages of data exploration project [5]

	Time to complete (percent of total)	Importance to success (percent of total)
1. Exploring the problem	10	15
2. Exploring the solution	9	14
3. Implementation specification	1	51
4. Data mining		
a. Data preparation	**60**	15
b. Data surveying	15	3
c. Data modelling	5	2

data understanding phase. In this stage the data pre-processing tool can help with data exploration and discovering for example duplicated or missing values, etc. Data pre-processing includes a lot of a tedious work. It can be simplified and shortened using specialised tools for data pre-processing.

2 SumatraTT

SumatraTT is a metadata-driven, platform independent, extensible, and universal data processing tool. These features have been achieved by building the tool as an interpreter of a transformation-oriented scripting language called SumatraScript. The Sumatra language is a fully interpreted language with Java-like syntax combining together data access, metadata access, and common programming constructions. Furthermore, it supports RAD (Rapid Application Development) technology by providing library of re-usable transformation templates. The principal schema of SumatraTT is shown in figure 1.

As it is shown in the figure, the central part of SumatraTT is the Metadata Repository module. Basically, the repository plays two roles. It is a central storage consisting of data sources descriptions and data transformation definitions. Moreover, the repository contains data objects interconnecting an abstract data access level in Sumatra interpreter with real-life data sources. This intermediated connection helps in unifying the data access to very different data sources (e.g. SQL-based data sources, plain text files, etc.). Such a unification makes the process of transformation script development easier and data source more independent. Moreover, it separates transformation "logic" from the data connection problems. In the case of very complicated data pre-processing task, development of a data transformation script can be very time consuming. The SumatraTT allows to speed-up this process by using re-applicable transformation templates. The idea of re-usable templates is based on a library of solved types of tasks. E.g., there is a data set containing time series and we need to calculate a statistical characterization of the data. If it is carried out for the first time the new template has to be developed. But next time, the statistical template will be ready. Thus, transformation script can be developed during a fraction of time required before.

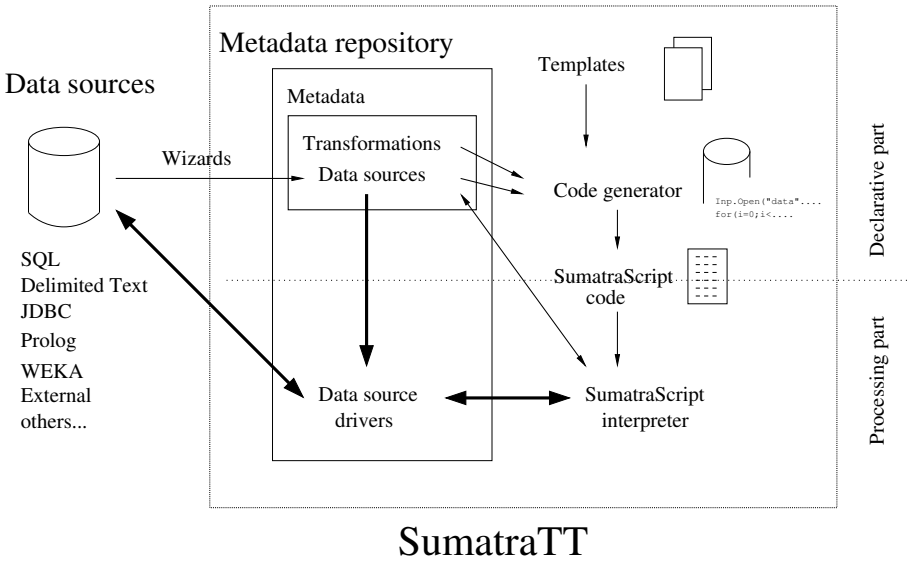


Fig. 1. Overview of SumatraTT

Every pre-processing task implemented using SumatraTT consists of a design and a run-time phases. It corresponds to the client-server architecture where the design phase means definition of all data sources and development of transformation scripts on the client side. In case of a typical user, who is an expert in data mining or data warehousing but who is not a programmer the design phase can be carried out using a graphical user interface. The GUI allows an interactive data definition and script development by simple clicking on wizards. On the other hand, the run-time phase corresponds to the script execution on the server side. From the user perspective, the execution can be invoked immediately or scheduled for running it later.

3 How SumatraTT Works

In the beginning, there is a data miner with data to be processed. Using SumatraTT there are several ways how to proceed.

The most straightforward way is to program the transformation task in SumatraScript. It is relatively difficult task. Although a unified data access and simplified programming are some of SumatraScript features that simplify transformation task development, almost the same result can be achieved using C++. Schema describing this approach is on figure 2.

The real benefit of using SumatraTT is making use of templates. Templates represent pre-programmed tasks already solved and generalized. They are written using meta-information, which is concretized at the moment of the final use. There exist several obligatory parameters, which expand the meta-information.

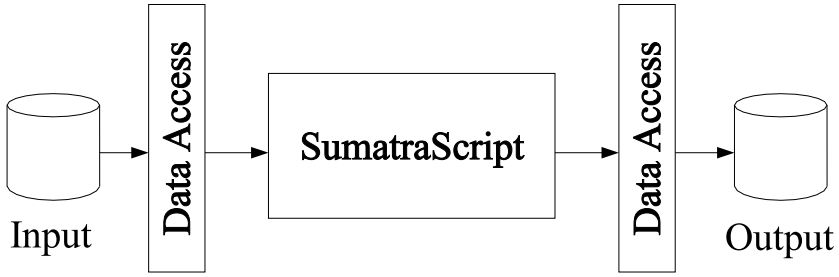


Fig. 2. Transformation written in SumatraScript

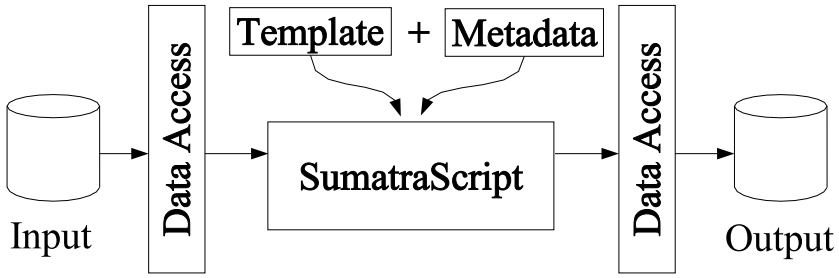


Fig. 3. Transformation with advance of template

Templates usually contain a lot of optional parameters for advanced features. A schema of using a template is in the figure 3.

In the second case SumatraTT combines the template and metadata and generates a complete SumatraScript code, which is processed in the same way as in the first case.

The best example of a template is the most frequently used one – `TransformTableCopy`. The main purpose of this template is to copy data from one data source into another. The arbitrary parameters are input and output data sources. At this moment, transformation is ready-to-use. It can move data from any (supported) format into another (e.g. from text file into database or from database into Prolog clauses, etc.). Format change is automatically achieved by different data source definition.

The main part of `TransformTableCopy` template consist of the following code (simplified). In metadata, the `From`, `To` parameters have to be specified to determine source and target data sources, the `Code` is optional.

```
from.FindFirst(); while(!from.Eof()) {
  // load data from data source to local variables
  ##DSvariablesLoadById(From,##param(From))
  // first of all, copy all matching fields
  ##DSCopyMatchingFields(From,##param(From),To,##param(To))
  // user-defined transform
  ##param(Code,)
  // store variables to data source
  ##DSvariablesSave(To,##param(To))
  To.PostRecord();
  From.NextRecord();
}
```

Optional parameters of `TransformTableCopy` template (which is much more developed than the previous example) extend SumatraTT's ability into a level of very powerful tool. There are parameters for creating the target data source (even if it is an SQL table), cleaning the target data source (removing all values) before copy, filtering, reporting problems during the run and others.

4 Case Study – Water Distribution Company

The case study is targeted to prediction of drinking water consumption in the Western Bohemian region. The distribution network starts from the manufacturing part, where the natural water quality is improved to be drinkable and then pumped to a primary water supply. A primary water supply distributes water to customers indirectly via storage reservoirs. Only a tiny amount of customers is supplied directly from a primary water supply. Each reservoir has its region where it works as the main source of drinking water. Water storage reservoirs are used as accumulators of drinking water to ensure fluent loading of water manufacturing part. The water is produced in a production unit (production = pumping from springs + cleaning). The best quality of water and most efficient production is achieved when producing a constant amount of water produced per time unit. Thus, the consumption peaks must be foreseen and the reservoirs must be pre-filled to contain enough water for covering following peaks. However, keeping non-necessary high level of water in water reservoirs is bad, as the water quality is affected by slow exchange of reservoir contents.

The reservoirs are automatically measured and controlled. Each of these reservoirs yields every 10 minutes measurements of inflow and outflow of water in cubic meters per second and a set of several design-dependent internal flows, which are used for control. Thus, a complete snapshot of process parameters is at disposal every 10 minutes.

From the beginning, the domain expert expected that water consumption is dependent on weather. Moreover, every reservoir distributes water to a different type of area (garden, block of flats, ...). Therefore a model for every reservoir has to be found. There were available the following data sets:

- Measurements of distribution network parameters (inflow, outflow, etc.) in water storage reservoirs. The data is stored in semicolon-separated plain text file where the first column represents date and time of the measurement and following 61 columns contain measured values.
- Distribution network description - connections and dependencies within the network (network configuration).
- Weather description

During the data pre-processing phase the following problems appeared:

- Data contains immediate values of inflow/outflow but the task is to predict water consumption, i.e. amount of water. The data has to be interpolated in an appropriate form.
- Each reservoir requires its special model. That is why the data has to be separated for each reservoir.
- The consumption also depends on weather in several previous days. That is why the data for a data mining tool have to contain several historical values in one record.

5 Problems Solved by SumatraTT

Generally, SumatraTT is able to solve almost every data pre-processing task. In the case of often-repeated tasks there is a corresponding template at disposal. Even if the template is not available, the task can be implemented by writing a transformation script. However, this is a time consuming way.

The advantage of using SumatraTT is in its simplicity, speed, and reliability of transformation design. SumatraTT's ability to solve tasks quickly is given by a **library of templates**. A set of basic templates is a part of the software distribution. These templates are able to solve wide range of common data pre-processing problems. Moreover, the library is still growing as new tasks are solved in a general way.

Every transformation is defined by a template to be applied and a **set of parameters** defined in **metadata**. The parameters determine the final behavior of the template. The list of currently supported tasks is mentioned in the next paragraphs. All these tasks can be easily implemented using a template from the standard library.

Usual data pre-processing tasks belongs to one of the following groups:

Standard tasks – supported by almost every transformation tool
Advanced tasks – very often in data mining practice but unfortunately not common in data pre-processing software

5.1 Standard Tasks

Currently, SumatraTT distribution [3] supports the following standard tasks:

Copy data Unfortunately, almost every data mining tool requires a special data format therefore the most frequent task is to change data format.

Formatting of values Especially text files different formats of data are used. The most common example of format issue is date and time processing. There are national formats of date, time, currencies. Changing data type and format is done automatically according to data source definition.

Calculation of a new attribute Adding a new or calculating derived attributes can be made by an expression in SumatraScript code (e.g. `out_SUM = in_Left+in_Right;`).

Data filtering Data filtering is defined by a logical condition. Optionally, there can exist a piece of code to solve more complicated situations.

Reporting The data pre-processing task can generate a text output or log errors and/or problems occurred during its run.

Visualization Data visualization is a very important function especially in the data understanding phase. It helps to uncover hidden features of the data. In SumatraTT, the visualization is available in two ways: directly in GUI using an interactive setup, or using external tool, e.g. `gnuplot`.

5.2 Advanced Tasks

There are many important issues that have to solved during the data pre-processing phase. They are often very and very time consuming. Unfortunately, currently available tools are missing their support. The following list contains problems that are usually very hard to solve by traditional tools and SumatraTT can solve them easily.

Split a data set It supports automatic splitting data into training and evaluation data sets with respect to the distribution of target classes. Moreover, the data set can be split into several data sets using a splitting condition.

Time series processing Data mining algorithms are mostly able to handle data sets only record by record. In the case of time series it happens very often that a value in time t somehow depends on values in time $t-1$, $t-2$, etc. Then data from several previous records have to be transformed into one record. The most of Sumatra TT templates support a parameter called `History` determining the depth of history to be held. These values can be accessed in a same way as normal fields are.

Moreover, application of the History parameter can help with deriving characteristic values of a subset of records. This feature was successfully verified in medical applications.

Contradictory records Sometimes data contains records, which have all values the same except class. Such records can be errors or can indicate that a set of attributes is not correct. SumatraTT contains a template where such records can be easily extracted.

Prolog facts export/import There is ready-made data source supporting export/import of data to/from Prolog facts. It is very useful in case of using ILP (Inductive Logic Programming) methods for data mining for details see [6] and [1].

6 Conclusion

There are many very sophisticated tools and algorithms for data mining as well as many good books in this area. They address lots of problems from different areas but only few of them describe how to prepare (pre-process) data for efficient analysis and achieving of usable results. This paper tries to point out some of the most often problems that have to be solved during data pre-processing phase. It describes their solution using a data pre-processing tool called SumatraTT.

SumatraTT is metadata-driven, platform independent, extensible, and universal data processing tool. These features have been achieved by building the tool as an interpreter of a transformation-oriented scripting language called SumatraScript. The Sumatra language is fully interpreted Java-like language combining together data access, metadata access, and common programming constructions. Furthermore, it supports RAD (Rapid Application Development) technology by the library of re-usable transformation templates.

The SumatraTT tool was verified on several data mining problems. One of them has been briefly described in this paper. It demonstrates a very common problems appearing during time series processing, e.g. using historical values, national formats of date and time, etc. Moreover, it shows complicated transformation of measured values into features suitable for data mining algorithms.

References

1. Petr Aubrecht, Filip Železný, Petr Mikšovský, and Olga Štěpánková. Progress in SumatraTT: ILP Connectivity and More New Features. In *Data Mining and Decision Support in Action! (subconference of Information Society IS 2001)*, volume 1, pages 107–110. Ljubljana : Institut Jozef Stefan, 2001.
2. Petr Aubrecht, Filip Železný, Petr Mikšovský, and Olga Štěpánková. Sumatratt: Towards a universal data preprocessor. In *Cybernetics and Systems 2002*, volume II, pages 818–823, Vienna, 2002. Austrian Society for Cybernetics Studies.
3. Czech Technical University in Prague. SumatraTT Official Home Page, 2001. http://krizik.felk.cvut.cz/Sumatra.
4. Katharina Morik. The Representation Race – Preprocessing for Handling Time Phenomena. In *ECML*, Lecture Notes in Computer Science, pages 4–19. Springer, 2000. Invited talk.
5. Dorian Pyle. *Data Preparation for Data Mining.* Morgan Kaufmann Publishers, Inc., San Francisco, CA, USA, 1999.
6. Filip Zelezny, Petr Aubrecht, and Petr Miksovsky. Connecting Sumatra to Aleph. In Christophe Giraud-Carrier, Nada Lavrač, Steve Moyle, and Branko Kavšek, editors, *Integrating Aspects of Data Mining, Decision Support and Meta-Learning: Internal SolEuNet Session*, pages 43–52. ECML/PKDD'01 workshop notes, September 2001.

Algebraic Specification of Database models

Martin Hnátek

Department of Computer Science and Engineering, Faculty of Electrical Engineering,
Czech Technical University,
Karlovo nám. 13, 121 35 Prague 2
xhnatek@fel.cvut.cz**

Abstract. Typical problem during the process of database reverse engineering (DBRE) is insufficient exactness of the original definitions of transformations among the database models at the different levels of abstraction (e.g. the conceptual model, logical model, implementation model). This paper uses algebraic notation tool, the OBJ3 language, to express possible database model and transformation specification. This approach tries to separate automatic work and the points of human decision more precisely.

1 Introduction

The term data reverse engineering (DRE) evolved from the more generic term reverse engineering. Reverse engineering is a process to achieve understanding of the structure and interrelationships of a subject system. It is the goal of reverse engineering to create representations that document the subject and facilitate our understanding it what it is, how it works, and how it does not work. As a process, reverse engineering can be applied to each of the three principal aspects of a system: data, process, and control. Data reverse engineering concentrates on the data aspect of the system that is the organization. It is a collection of methods and tools to help an organization determine the structure, function, and meaning of its data. In case of data intensive applications where database system is used, the term database reverse engineering is applied. DRE has also been defined in several ways, for example as "the use of structured techniques to re-constitute the data assets of an existing system". For a deeper look at this definition see [1]. Reverse engineering and design recovery has been clearly and neatly defined by Elliot Chikofsky in [2].

Basically, DRE is seldom a goal of some project itself, but often is the first step in a broader engineering project. Here follow some of the most frequent objectives of DRE.

** This work has been partially supported by the research program no. MSM 212300014 "Research in the Area of Information Technologies and Communications" of the Czech Technical University in Prague (sponsored by the Ministry of Education, Youth and Sports of the Czech Republic.

- *Data extraction/conversion*: In some situations, the only component to salvage when abandoning a legacy system is its database. The data have to be converted into another format, which requires some knowledge on its physical and semantic characteristics. On the other hand, most data warehouses are filled with aggregated data extracted from corporate databases. This transfer requires a deep understanding of the physical data structures, to write the extraction routines, and of their semantics, to interpret them correctly.
- *Quality assessment*: Analyzing the code and the data structures of a system in some detail can bring useful hints about the quality of this system, and about the way it was developed. M. Blaha observed that assessing the quality of a vendor software database through database reverse engineering techniques is a good way to evaluate the quality of the whole system.
- *Data administration*: database reverse engineering is also required when one develops a data administration function that has to know and record the description of all the information resources of the organization.
- *Component reuse*: In emerging system architectures, database reverse engineering allows developers to identify, extract and wrap legacy functional and data components in order to integrate them in new systems, generally through ORB technologies.

Extensive list of reasons for practicing DRE has been presented in [8]. In a wider look, data reverse engineering is a special subset of reverse engineering and reverse engineering is a part of software maintenance, thus necessity of DRE is derived from inevitability of software maintenance. Particular forms of software maintenance and reasons why it can not be shifted off is analyzed in [10].

Data reverse engineering grew up through two different (and for the most part exclusive) communities:

1. the database community,
2. the software engineering community.

The way DRE was performed and which aspects of DRE were emphasized differed as much as the focus of the two communities differed. The database community always used own terminology and methodologies. The software engineering community perceived the data reverse engineering as a special aspect of the whole reverse engineering process.

Over the years, the research and publications in DRE by both communities has been mainly in three areas:

- DRE translation and methodologies algorithms,
- DRE tools,
- The DRE of specific applications and experiences in DRE.

This paper concentrates on formal model description, together with transformational approach elaboration. The database community constructed number of models describing different levels of database abstraction, starting at conceptual

schema, going through logical schema, ending with implementation. These models help designing database and they correspond engineering tasks very well. However, the models are not suitable for reverse engineering purposes which requires other properties of description. The main goal of data reverse engineering models is to facilitate the engineer in several ways. High degree of model equivalences is required to express inter-schema mappings appropriately. Feasible description offers mathematical characterization of transformations and brings possibility of precise verification and validation. Also, systematic approach to parameterization must by used enabling model conformation for the particular task. On the other hand, no process is substitute for expertise of software engineer, so the model specification must provide the possibility to precisely state the moments of human knowledge and decision utilization.

There are several formal approaches that can be utilized with various features and applicability. The first way presents individual model creation adjusted to data reverse engineering processes. The DB-MAIN approach is such an example [7]. The second approach applies algebraic specifications for model description and relevant algebraic operations for transformations. The algebraic approach is employed in this paper. The third alternative considers type theories.

2 A Brief Summary of Algebra Variants

Oversimplified, the main idea of algebraic specifications is the description of the modeled system using algebraic structures. All the properties of algebraic structures are exactly mathematically defined in a number of publication, for example [4] or [12] (in Czech). General (also called universal) algebra offered basic concepts (such as congruence, variety, and free algebra) and basic results (including completeness and variety theorems). Anyway, ordinary unsorted logic allows the situation where anything can be applied to anything, thus unsorted logic is too permissive. Later, many sorted algebra (MSA) has been studied with better results using sorts, for example in automatic theorem proving this can greatly reduce the search space. Although many sorted algebra has been quite successful for the theory of abstract data types, it can produce some very awkward code in practice, primarily due to difficulties in handling erroneous expressions, such as dividing by zero in the rationals, or taking the top of an empty stack. In fact, there is no really satisfying way to define either rationals or stacks with (unconditional) MSA. Traditional many sorted logic is too restrictive, because it does not support overloaded operation symbols, such as .+. for integer, rational, and complex numbers. Strictly speaking, an expression like (-21 / -3)! does not parse (assuming that factorial only applies to natural numbers), because (-21 / -3) looks to the parser like a rational rather than a natural. As a solution, the use of sorted sets (also called indexed families) for MSA was introduced by Goguen in lectures at the University of Chicago, and first appeared in print in [3].

Sorted sets allow a simpler notation than alternative approaches, and also allow overloading. However, overloading only reveals its full potential in order

sorted algebra (OSA). OSA is designed to handle cases where things of one sort are also of another sort (e.g., all natural numbers are also rational numbers), and where operators or expressions may have several different sorts. The essence of order sorted algebra is to provide a subsort partial ordering among sorts, and to interpret it semantically as subset inclusion among the carriers of models; for example, given Nat and Rat and signalizing that $M_{Nat} \subseteq M_{Rat}$, where M is a model, and M_s is its set of elements of sort s. OSA also supports multiple inheritance, in the sense that a given sort may have more than one distinct supersort. Two pleasant facts are that OSA is only slightly more difficult than many sorted algebra, and that essentially all results generalize without difficulty from the many sorted to the order sorted case. OSA presents the basis for the OBJ language.

Ongoing research brought alternative approaches, for example universe OSA, unified algebra, hidden sorted algebra. There is a comprehensive survey of algebras, mostly aimed at order sorted algebras, in [6].

3 The OBJ3 Language

OBJ is a wide spectrum first-order functional language that is rigorously based on (order sorted) equational logic and parameterized programming, supporting a declarative style. The following text is meant as a recapitulation of OBJ3 features, giving a list of typical ones, and commenting only the most utilized. A solid introduction to OBJ is included in [5], also available in PostScript from [11].

1. OBJ is based on OSA. OSA provides a notion of subsort that rigorously supports multiple inheritance, exception handling and overloading.
2. Two kinds of modules: objects (which are not very closely related to objects in the sense of object-oriented programming) and theories. Objects encapsulate executable code, and in particular define abstract data types by initial algebra semantics, and (loose) theories are designed to specify both syntactic and semantic properties of modules.
3. Parametrized programming. Parametrized programming gives powerful support for design, verification, reuse, and maintenance.
4. Executable code consists of equations that are interpreted as rewrite rules. OBJ provides some extended features, such as rewriting modulo associative, commutative and/or identity equations, as well as user-definable evaluation strategies that allow lazy, eager, and mixed evaluation strategies on an operator-by-operator basis. In addition, OBJ3 supports the application of equations one at a time, either forwards or backwards; this is needed for equational theorem proving.
5. Mixfix syntax. OBJ allows users to define any syntax for operators, including prefix (for example `top_`), postfix (for example `_squared`), infix (for example `_+_`), and most generally, mixfix (for example `if_then_else_fi`).
6. OBJ3 extensions. OBJ3 objects can encapsulate Lisp code, e.g., to provide efficient built-in data types, or to augment the system with new capabilities.

7. Built-in modules. OBJ is sufficiently powerful to define any desired data type, but building in the most frequently used data types can make a great difference in efficiency and convenience. OBJ3 has predefined objects such as BOOL, IDENTICAL, NAT, INT, RAT, FLOAT, QID, QIDL, and ID, plus the parameterized tuple objects, the theory TRIV, etc.

OSA provides sufficient expressiveness, while still banishing truly meaningless expressions. It allows to separate logically and intuitively distinct concepts and, when the notion of subsort is added, to support multiple inheritance, overloading (a form of subsort polymorphism), coercions, multiple representations, and error handling, without the confusion, and lack of semantics, found in many programming languages. In particular, overloading can allow users to write simpler expressions, because context can often determine which possibility is intended.

OBJ offers powerful means directed to parametrized programming. Each kind of module (object or theory) can be parameterized, where actual parameters may be modules. For parameter instantiation, a view binds the formal entities in an interface theory to actual entities in a module, and also asserts that the target module satisfies the semantic conditions of the interface theory. This kind of module composition is, in practice, more powerful than the purely functional composition of traditional functional programming, because a single module instantiation can compose together many different functions all at once, in complex ways.

Typical OBJ3 programming session consists of designing an object, witch possibility of using other models as parameters, and trying reduction of some patterns. Here is a recall of initial object syntax:

```
obj <ModId> is
  sort <SortId> .                          *** sort declaration
  op <OpForm> : <SortList> -> <Sort> .  *** operation declaration
  var <VarId> : <Sort> .                   *** variable declaration
  eq <Term> = <Term> .                     *** ordinary equation
endo
```

Reduction is declared by

```
reduce [ in <ModExp> :] <Term>
```

Here is the simplest illustration of reduction:

```
reduce in NAT : 1 + 2 .
```

The introduction to OBJ [5] includes number of examples, as well as OBJ3 syntax grammar.

4 Specifying Database Models: The Several Object Approach

The following lines attempt to sketch database model specification, emphasizing debate of rising aspects and offering some examples. Source pieces in the text are exemplary patterns only. Complete and executable code is available online [9].

Specification of database models using OBJ3 is motivated in cumulative improvements. First task presents simply expressing models utilizing OBJ3 objects. Next phase tries to define forward and reverse process, evaluating equivalences (or differences) of particular operations among them. Final phase presents variety of case studies interpretation showing practicability of selected approach.

The process of model specification is influenced by a number of decisions.

- The scale of parameterization. Some situations allow to select either an extension of already existing object or parameterization of the feature.
- Database types representation. Obviously, there is a possibility of the OBJ language means usage (e.g. database type would be an OBJ object), or the approach of specifying a sort for database type, together with corresponding operations.
- The points of expected inconsistencies. Incorrect design is indicated as distinct request in the process of transformation which must be supported.
- Different initial object structure at different levels of abstraction. First approach tries unifying all the models into one generic model from the very beginning. The second one builds particular object for particular level of abstraction and consequently formalizes the transformations.

Obviously, during the process of object completion, the approaches of greater hierarchy versus one big object pervade in several ways, but the initial state is still important.

In this paper, the first approach is showed, focusing on conceptual to logical model migration, using the ER model and the relational model terminology in examples.

Supposing already existing type object that specifies types which will be used, the attribute description can be specified like this:

```
obj ATTRIB is
  sort Attrib .
  protecting QID .
  protecting TYPE .
  op (_:_) : Id Type -> Attrib .
  op name : Attrib -> Id .
  op type : Attrib -> Type .
  var A : Attrib .
  var I : Id .
  var T : Type .
  eq name(I : T) = I .
```

```
  eq type(I : T) = T .
endo
```

Similar ideas lead to entity and association (n-ary relation) specification. Attribute realization involve name and type completion, entity is created specifying the name together with the list of attributes. Association joins together two or more entities, adding arity.

The next step presents forward and reverse object specification. In both cases it is desirable to distinguish two kinds of transformations: type transformation and structure transformation. The important observation follows equivalences or differences in the contradictory processes. Decision of symmetry is often determined by semantic inconsistency which should be designated by the engineer, thus the problem of loose design is exposed in different local transformation specification. The source code on [9] presents sample reductions showing typical problematic reverse steps.

5 Conclusions and future work

Algebraic specification of database models using OBJ3 language lined up several ideas in the process of model transformations. Different aspects of model transformation has been separated and the points of user (or other outside) decisions has been outlined. Anyway, a lot of issues has to be improved. All considerations should be generalized to emphasize fundamentals of model at the selected level of abstraction, and to leave over more specific details. This means greater parameterization of objects in OBJ3 lingo. On the other hand, models at the implementation level has to be elaborated, to reveal practical (but theoretically undesirable) features of data definition languages. A lot of steps needs to be done in the benefit of user (reverse engineer) interaction. At least, notation on a higher level for human decision definitions must be provided. Ongoing research should cover the hole problem of database model transformations, together with concurrence links to the rest of reverse engineering processes settlement.

References

1. P. Aiken. *Data Reverse Engineering: Slaying the Legacy Dragon.* McGraw-Hill, 1996.
2. E. Chikofsky, and J. Cross. *Reverse Engineering and Design Recovery: A Taxonomy.* IEEE Software, January 1990, 7(1):13-17.
3. J. Goguen. Semantics of computation. In Ernest G. Manes, editor, *Proceedings, First International Symposium on Category Theory Applied to Computation and Control*, pages 234-249. University of Massachusetts at Amherst, 1974.
4. J. Goguen. *Order sorted algebra.* Technical Report 14, UCLA Computer Science Department, 1978. Semantics and Theory of Computation Series.
5. J. Goguen, T. Winkler, J. Meseguer, K. Futatsugi, and J.P. Jouannaud. *Introducing OBJ.* SRI Computer Science Laboratory Technical Report SRI-CSL-92-03, Menlo Park 1992.

6. J. Goguen, and R. Diaconescu. An Oxford survey of order sorted algebra. Mathematical Structures in Computer Science, 4:363-392, 1994.
7. J-L. Hainaut, V. Englebert, J. Henrard, J-M. Hick, and D. Roland. Evolution of database Applications: the DB-MAIN Approach. In *Proceedings of the thirteenth Conf. on the Entity-Relationship Approach*, Manchester, December 1994.
8. J-L. Hainaut, J. Henrard, J-M. Hick, V. Englebert, and D. Roland. The Nature of Data Reverse Engineering. *Data Reverse Engineering Workshop*, EuroRef, Seventh Reengineering Forum, ReengineeringWeek 2000, Zurich, Switzerland, March 2000.
9. *Research projects* [online]. Department of Computer Science and Engineering, Faculty of Electrical Engineering, Czech Technical University, 20 March 2003; 15:50:23 [cited 20 March 2003]. Available from: ⟨http://cs.felk.cvut.cz/ xhnatek/research/⟩
10. K. Lano, and H. Haughton. *Reverse engineering and software maintenance*. McGraw-Hill International Series in Software Engineering, 1993.
11. *OBJ Family: OBJ3 CafeOBJ Maude Kumo FOOPS Eqlog* [online]. UCSD Department of Computer Science and Engineering - CSE Home, 24 October 2002; 02:49:22 [cited 10 March 2003]. Available from: ⟨http://www.cs.ucsd.edu/users/goguen/sys/obj.html⟩
12. K. Richta, and J. Velebil. *Sémantika programovacích jazyků*. Karolinum, 1997.

Benchmarking the Multidimensional Approach for Term Searching

Jiří Dvorský, Michal Krátký, Tomáš Skopal, and Václav Snášel

Department of Computer Science, VŠB-Technical University of Ostrava, Czech Republic

Abstract. The area of computer science dealing with searching and processing collections of multimedial documents is known as Information Retrieval. In information retrieval systems the text documents are characterized with terms. For effective retrieval the terms must be stored in appropriate data structures. Many term searching approaches (e.g. inverted list) are able to process only the simple queries. We can use these approaches for finding out whether the term is or is not stored in the term database. But non-trivial term searching (e.g. according regular expressions) is hardly feasible or even impossible. In this paper, we introduce a multidimensional approach for non-trivial term searching.

1 Introduction

Area, which deals with processing and searching in multimedia documents, is called *Information Retrieval* [14,1,8]. Textual documents, in the information retrieval system, are characterized by terms. In general the term can be word or multi word phrase. It is necessary to choose appropriate data structure for searching of terms. Purpose of term indexing is dependent on used model (e.g. *Boolean* or *Vector model* [14]). Methods of terms storage developed in past (such as inverted file) allow simple querying on set of terms only. In most cases we can only decide if particular document is characterized by given term or is not. More complex searching such as searching of the terms given by regular expression is very difficult or even impossible. We propose multidimensional approach for non-trivial searching of terms in our paper.

The inverted file is one of the approaches of terms storage that characterize particular document. The inverted file consists of alphabetically sorted terms and for all of these terms there is a list of documents characterized by the term. The inverted file can be represented as B-tree [16]. Searching in B-tree can be done in logarithmic time with respect to number of terms in tree. So, the inverted file solve problem how to find documents for exactly given term.

This approach does not allow retrieving documents for queries, which consist of regular expressions. For example regular expression `"*Heller"` means retrieving all of documents in which terms `"Joseph Heller"` or `"Petr Heller"` appear. Searching algorithm for inverted file, implemented as B-tree, that use regular expression is described in [8]. This approach is very efficient for query

© J. Pokorný, V. Snášel (Eds.): Dateso 2003, pp. 71–82, ISBN 80-248-0330-5.
VŠB – Technical University of Ostrava, Dept. of Computer Science, 2003.

processing of `<string>*` type. The terms could be ordered from last symbol for efficient processing of `*<string>` type query. But it is unreal to create special index for each query type.

We introduce the multidimensional approach for efficient query processing defined by regular expression over single index. Multidimensional approach for non-trivial term searching will be presented in Section 2. Great emphasis will be laid to construction of queries based on user's demand. The multidimensional approach utilizes structures for indexing multidimensional vector spaces. One of these structures - UB-tree - will be described in Section 3. The experimental results will be presented in Section 4. Conclusions summarize the contribution of this paper and give outlook to the future.

2 Multidimensional approach for non-trivial term querying

Structures for indexing of *discrete vector spaces* [5] are used in multidimensional approach for non-trivial term querying. These structures allow us to consider terms as points in n-dimensional vector space. UB-trees [2] and R-trees [9,4] belong to such kind of structures. Retrieving terms from query in this case is transformed to finding points in given n-dimensional box. Such kind of queries is called *multidimensional range query*. It is necessary to transform term to n-dimensional point for effective storage and searching in index structures of vector spaces. The transformation is often called *feature extraction* [5], i.e. relevant information is obtained from indexed entity, and the obtained piece of information can be processed for indexing.

Definition 1 (n-dimensional point representing term).
Let $t = c_0, c_1, \ldots, c_{n-2}, c_{n-1}$ be a term of the length n, where $c_i \in \mathcal{A}$ is character, $0 \leq i < n$. n-dimensional point representing the term t is defined as $p_t = (code(c_0), code(c_1), \ldots code(c_{n-1}))$, where $code : \mathcal{A} \rightarrow \{0,1\}^m$ is function, which encodes character to binary number of the length m.

It is necessary to choose maximal length of term n, and terms of the length l, where $l < n$, will have coordinates $p_{term_i}, l \leq i < n$ equal to zero. Dimension of space Ω is equal to n and cardinality of domains $|D_i| = 2^m, 1 \leq i < n$. The dimension of space is in the most cases smaller than 30, which follows from lengths of words in natural languages. The cardinality of domains is 256, when ASCII coding is used. The number of dimension is very large, and effectiveness of the algorithms could be affected by *curse of dimensionality* [5]. On the other hand cardinality of domains is relatively small.

Let $max_d = |D_i| - 1 = 2^m - 1$ (i.e. value that can be expressed as m bit number) be a maximal value of domain i. In case of ASCII coding $max_d = 2^8 - 1 = 255$. The term, that is transformed to point in n-dimensional vector space, is inserted into multidimensional data structure, e.g. UB-tree. Queries are then performed on the multidimensional structure.

2.1 Construction of queries

A task of finding terms in multidimensional approach is transformed to geometrical task of finding points in multidimensional cube. This n-dimensional cube will be called query box. The box can be defined by two n-dimensional points. When user enters his/her query, we must be able to create particular multidimensional query box (i.e. two points determining box must be defined) and in data structure search for points that belong to the box. The points obtained from data structure should be decoded to terms (decoding is done by application of function $code^{-1}$). A list of decoded terms is returned to the user as result.

Example 1 (The construction of query box for regular expression query.). Let's take terms `fan`, `fun` and `far`. Corresponding points for these words are:

$$\begin{aligned} p_{fan} &= (code(f), code(e), code(n)) = (102, 97, 110) \\ p_{fun} &= (code(f), code(u), code(n)) = (102, 117, 110) \\ p_{far} &= (code(f), code(a), code(r)) = (102, 97, 114) \end{aligned}$$

Function *code* computes ASCII encoding in this case, the cardinality of domains is $|D_i| = 256, 1 \leq i \leq 3$. The dimension n of the space has to be 3 due to easy visualization (see Figure 1).

Let's take two regular expression query `(f*n)` and `(f*)`. The range query boxes will be constructed for given expressions. The 3-dimensional box

$$(code(f), 0, code(n)) \times (code(f), max_d, code(n))$$

is defined for the first of expressions. The constructed box can be seen in Figure 1. Every point of the box has the first and the third coordinates constant, only the second coordinate is drawn from interval $\langle 0, max_d \rangle$. Results of the query represented by the query box 1 are `fan` and `fun` only. We don't think the query box retrieving the term `fn` now.

The second query defines a query box

$$(code(f), 0, 0) \times (code(f), max_d, max_d)$$

Resulting query box can be seen in Figure 1. The results for the query box are terms `fan, fun` and `far`.

3 The multidimensional data structures

The multidimensional data structures are structures, which index the vector (for example R-tree [9], UB-tree [2]) or metric spaces (M-tree [6]). We have used the UB-tree for the multidimensional approach for term indexing. The reason of UB-tree usage is very good resistance again *curse of dimensionality* [5]. The space dimension could be up 30 in this case.

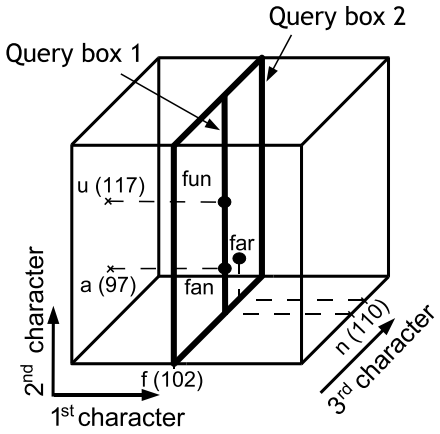


Fig. 1. 3-dimensional space containing points representing terms and query boxes for retrieving terms defined by regular expressions `(f*n)` and `(f*)`.

3.1 The Universal B-tree (UB-tree)

The *Universal B-tree* (*UB-tree*) belongs to methods for accessing multidimensional data in a multidimensional space. This access method was designed in [2]. The extensive source of information about UB-tree and multidimensional indexing is [12,3].

The idea of the UB-tree is based on the *Z-ordering* and the *B-tree*. The *B-tree* is balanced, persistent, paging tree, which provides logarithmic complexities for basic operations and minimal overhead over inserting and deleting of indexed tuples/points. A page utilisation of 50% is guaranteed. It is necessary to establish an ordering at points of n-dimensional space for using *B-tree* data structure. *Z-ordering* (*Z-address*) is used for the UB-trees.

Z-address

Definition 2 (Z-address).
Let Ω be an n-dimensional space. For tuple $\mathcal{O} \in \Omega$ with n attributes and binary representation attribute value $A_i = A_{i,s-1}A_{i,s-2}\ldots A_{i,0}$, where $1 \leq i \leq n$, it is defined function $Z(\mathcal{O})$:

$$Z(\mathcal{O}) = \sum_{j=0}^{s-1}\sum_{i=1}^{n} A_{i,j}2^{jn+i-1}$$

The function calculates for all points of n-dimensional space their Z-addresses. Because calculation of the Z-address is very frequent operation upon

using the UB-tree, it is important the *algorithm of bit-interleaving* with linear complexity to exist for its calculation. We can see Z-addresses for 2-dimensional space 8×8 in Figure 2. Thus the Z-ordering transforms the n-dimensional space into a one dimensional space and establishes ordering at points of n-dimensional space. If we are calculating the Z-addresses for all points of n-dimensional space Ω we get the *Z-curve* filling the entire space Ω (see Figure 2b).

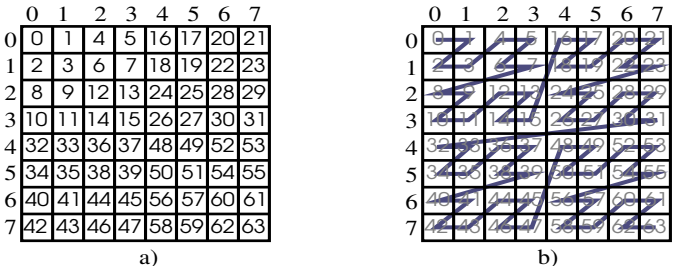


Fig. 2. a) The Z-addresses of all points 2-dimensional space 8×8. b) The Z-curve filling the entire 2-dimensional space 8×8.

Z-region

The UB-tree establishes *Z-regions* for clustering spatial neighbours onto disk pages. The Z-regions allow an effective disk access upon getting spatial neighbours and so an efficient processing of the multidimensional range queries. The *Z-region [α:β]* is defined as the space covered by interval $\langle \alpha, \beta \rangle$ on Z-curve. There is the Z-region [3:18] in 2-dimensional space 8×8 in Figure 3a). The entire space can be partitioned into the Z-regions (see Figure 3b). The Z-regions define a totally ordered disjunctive partitioning of n-dimensional space. Tuples from the Z-region are stored into a single B-tree page. The Z-regions are mapped onto disk pages. Thus the usage of Z-regions ensures the minimal number of disk accesses upon getting spatial neighbours.

The UB-tree is based on a B^+-tree. The B^+-tree is a B-tree with all keys considered in leafs only. The way how to store indexed tuples and Z-regions in UB-tree is depicted in Figure 4.

Range query

Basic query (together with point query) is *range query*, the query for finding of points in given n-dimensional hyper box. The exponential algorithm (with dimension) is shown in [2], in [12,13] is very vaguely presented the linear algorithm. For that reason, authors of this paper have developed their own algorithm called *DRU range query algorithm* [15].

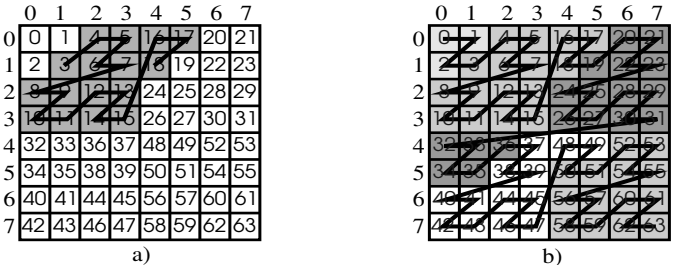


Fig. 3. a) The Z-region [3:18] in 2-dimensional space 8×8. b) The partition of 2-dimensional space 8×8 defined by Z-regions [0:2],[3:18],[19:36],[37:49],[50:63].

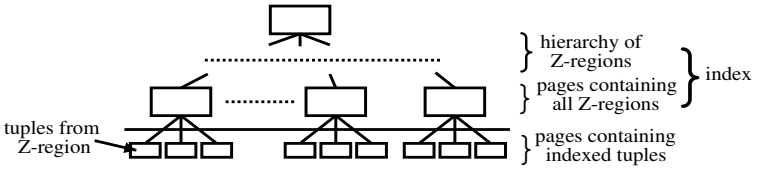


Fig. 4. The storage for indexed tuples and Z-regions in UB-tree.

4 Experimental Results

For a set of tests we have chosen a real dataset – the terms extracted from the TREC's `Los Angeles Times` collection which includes over 130,000 of newspaper articles and about 662,000 unique terms. The number of tuples in indices is growing with growing dimensionality (terms length respectively) but for $n \geq 17$ the number of tuples is almost constant (for example see Figure 5(b) – Whole terms count). Short list of important characteristics of created UB-trees can be found in Table 1. For each of UB-trees corresponding B-tree was constructed i.e. B-tree that contains the same set of terms. Characteristics of obtained B-trees can be found in Table 2. The native implementation of UB-tree with own range query algorithm [15] was used. Tests were executed at computer Intel Pentium®4 2.4Ghz, 512MB DDR333, Windows XP.

Table 1. UB-tree characteristics

$\|D_i\|$	2^8	dimensions	10–30
tuples	408,962–662,000	tree height	4
nodes	22,068–37,824	Z-regions	20,626–35,341
node capacity	26	utilization	71.3–71.2%
node size	490–1030B	index file	10.3–37.2MB

Table 2. B-tree characteristics

tree height	5	items	408,962–662,000
nodes	19,622–38,605	index file	26–52 MB
node capacity	26	utilization	80–66%

4.1 Performed Tests

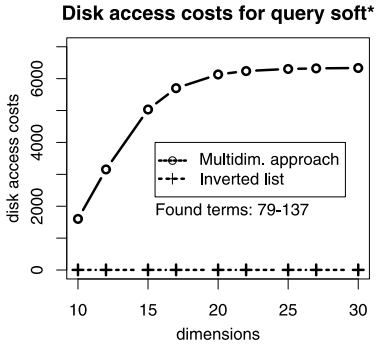
Several typical queries, defined by regular expressions: `soft*`, `*soft*`, `*soft`, were tested on UB-trees respectively B-trees (inverted list). In all cases disk access cost, number of found terms, number of all compared terms, and overall execution time were observed with respect to increasing length of terms. The range query was constructed for each regular expression in case of UB-tree. For single query defined by a regular expression it is necessary to process generally more range queries. The range query selectivity have been 12.8–6.2 on average (with growing dimension the count of queries return the empty result extended). A finite automaton corresponding to regular expression was constructed to perform searching on B-tree.

4.2 Experimental Results – evaluation

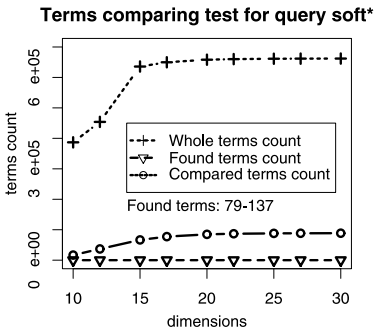
Query `soft*`

B-tree (inverted list) indicates itself as better for query `soft*` (query type `<string>*`). The disk access counts is minimal (proportional to tree height). Reason is we are capably to determine the first and last searched terms. In other words we are able to define the order for this query type homomorphic with terms order in B-tree. For multidimensional approach it is necessary to process only one range query. The terms count matched to query were found 79–137 (in dependency at maximal term length).

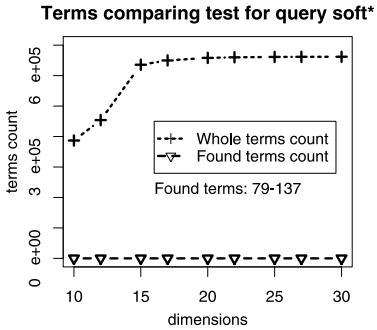
In Figure 5(a) we see the disk access counts (DAC) for inverted list and multidimensional approach. We see the DAC is positively lesser for inverted list. The Figures 5(b) and (c) show the compared terms count (CTC) essential for this query processing in multidimensional approach respectively inverted list. The count is far lesser than whole terms count, sure for inverted list is count only about one higher than found terms count (FTC). This fact corresponds to time query processing in Figure 5(d). This query processing is ideal for inverted list but we see the query processing is very efficiently in multidimensional approach too.



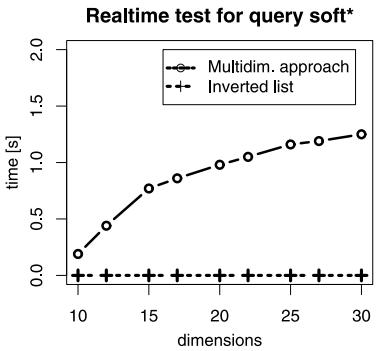
(a) Disk access costs.



(b) Terms count comparing test – UB-tree.



(c) Terms count comparing test – B-tree.

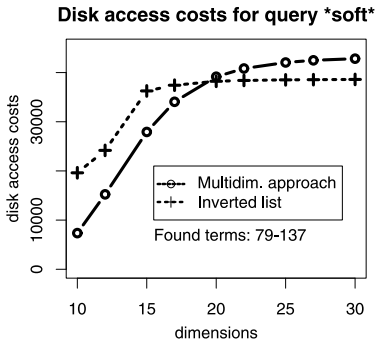


(d) Realtime test.

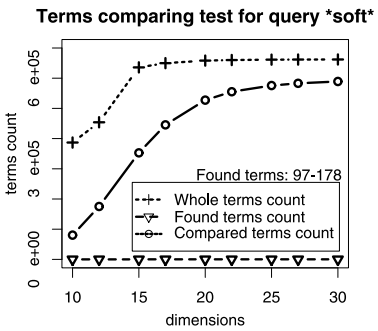
Fig. 5. Experimental results for query `soft*`.

Query `*soft*`

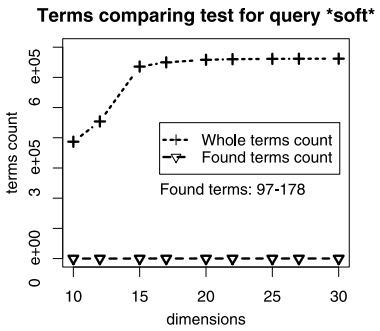
The B-tree proves the extremaly behaviour from compared terms count point of view. The CTC is about one higher than the found terms count for query `soft*` (last compared term is first non-matching term) but for other query (`*soft*` and `*soft` for example) is conversely necessary to search the whole B-tree. In that case the compared terms count blends with the whole terms count (see Figure 6(c)). The UB-tree compares the more terms than is the found terms count but the CTC is always lesser than whole terms count (see Figure 6(b)). For this query we see the CTC nears to the whole terms count for dimension 30 sure the senseful maximal term length is about 20.



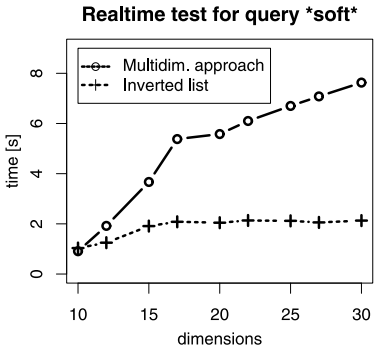
(a) Disk access costs.



(b) Terms count comparing test – UB-tree.



(c) Terms count comparing test – B-tree.



(d) Realtime test.

Fig. 6. Experimental results for query `*soft*`.

We must to process the $n - len(string) + 1$ (in that case n-3) of range queries in multidimensional approach for process this query. The found terms count is 97–178 (in dependency at maximal term length – space dimension).

Query `*soft`

The number of terms appropriate to this query is 16–24. For this query it is necessary to compare the all terms (see Figure 7(c)). In case of multidimensional approach it is essential to perform the $n - 3$ of range queries but the compared terms count is far lesser from the whole terms count (see Figure 7(b)). This fact is reflected in query process time for multidimensional approach (see Figure 7(d)). The results of process this query suit about very well efficiency of the multidimensional approach.

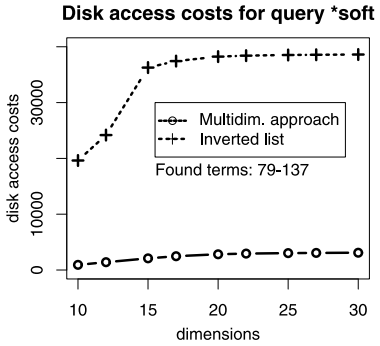
5 Conclusion

The multidimensional approach for term indexing offers much more capabilities for terms querying than current approaches (inverted list for example). We presented this approach for processing of queries defined by regular expression. But this approach is suitable for processing of similarity queries for Hamming or Levenshtein metrics [11,10]. Some queries are very efficiently processed by inverted list but for great amount of queries it is necessary to compare all terms. For processing of single query it is needful to execute generally more of range queries in multidimensional approach but the effectivity is very good even for higher dimension (maximal term length). The content of our future work is improve the properties of multidimensional approach.

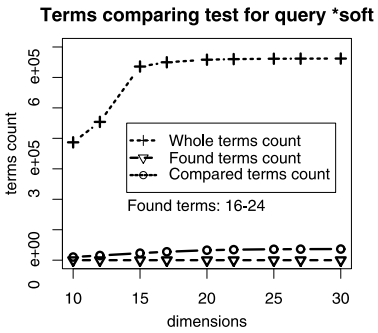
The experimental result shows that effectivity slims the increasing dimension. Therefore we will attempt any reduction of term space dimension or the thicking of the space (because the whole terms count is very small with comparing to volume of term space). Next open question is quality of elected clustering respectively the finding of suitable term clustering. In our future work we want to use data structures for metric space indexing (for example M-tree [6]) or to compare our approach to exist metric terms indexing approach (for example [7]). But these structures knows to index the objects only for one metric. Likewise we want try to find the method for defining of hyper box for general regular expressions with possibility of usage the hyper box geometric optimisations (as unification or intersection).

References

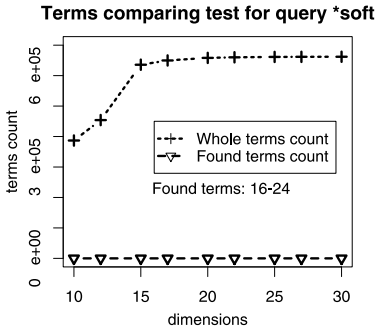
1. R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval*. Addison Wesley, New York, 1999.
2. R. Bayer. The Universal B-Tree for multidimensional indexing: General Concepts. In *Proceedings of World-Wide Computing and its Applications'97, WWCA'97, Tsukuba, Japan*, 1997.



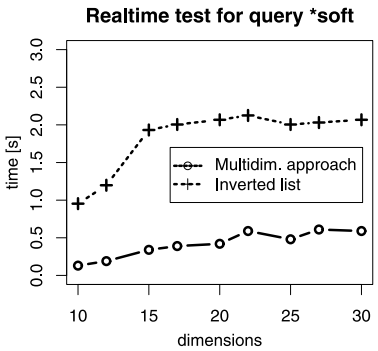
(a) Disk access costs.



(b) Terms count comparing test – UB-tree.



(c) Terms count comparing test – B-tree.



(d) Realtime test.

Fig. 7. Experimental results for query `*soft`.

3. Bayer Rudolf et al. *The project MISTRAL (Multidimensional Indexes for Storage and for the Relational Algebra).* `http://mistral.in.tum.de`, 1999.
4. N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger. The R*-tree: An effcient and robust access method for points and rectangles. In *Proceedings of the 1990 ACM SIGMOD International Conference on Management of Data*, pages 322–331, 1990.
5. C. Böhm, S. Berchtold, and D. Keim. Searching in High-dimensional Spaces – Index Structures for Improving the Performance Of Multimedia Databases. *ACM*, 2002.
6. P. Ciaccia, M. Pattela, and P. Zezula. M-tree: An Efficient Access Method for Similarity Search in Metric Spaces. In *Proceedings of 23rd International Conference on VLDB*, pages 426–435, 1997.
7. V. Dohnal, C. Gennaro, and P. Zezula. A Metric Index for Approximate Text Management. In *Proceedings of IASTED International Conference Information Systems and Database – ISDB 2002*, 2002.
8. W. Frakes and R. Baeza-Yates. *Information Retrieval: Data Structures and Algorithms.* Prentice-Hall, 1992.
9. A. Guttman. R-Trees: A Dynamic Index Structure for Spatial Searching. In *Proceedings of ACM SIGMOD 1984, Annual Meeting, Boston, USA*, pages 47–57. ACM Press, June 1984.
10. J. Holub. *Simulation of Nondeterministic Finite Automata in Pattern Matching, Ph.D. thesis.* Czech Technical University, Prague, `http://cs.felk.cvut.cz/~holub/`, 2000.
11. I. Levenshtein. Binary codes capable of correcting deletions, insertions and reversals. *Soviet Physics-Doklady 10*, pages 707–710, 1966.
12. V. Markl. Mistral: Processing Relational Queries using a Multidimensional Access Technique. In *Ph.D. thesis, Technical Univerity München, German*, `http://mistral.in.tum.de/results/publications/Mar99.pdf`, 1999.
13. F. Ramsak, V. Markl, R. F. R., M. Zirkel, K. Elhardt, and R. Bayer. Integrating the UB-tree into a Database System Kernel. In *Proceedings of 26th VLDB International Conference.* Morgan Kaufmann, 2000.
14. G. Salton and M. McGill. *Introduction to Modern Information Retrieval.* McGraw Hill Publications, 1st edition, 1983.
15. T. Skopal, M. Krátký, V. Snášel, and J. Pokorný. On Range Queries in Universal B-trees. In *Submitted at VLDB International Conference*, 2003.
16. N. Wirth. *Algorithms and Data Structures.* Prentice Hall, 1984.

Benchmarking the UB-tree

Michal Krátký, Tomáš Skopal

Department of Computer Science, VŠB–Technical University of Ostrava,
tř. 17. listopadu 15, Ostrava, Czech Republic
michal.kratky@vsb.cz, tomas.skopal@vsb.cz

Abstract. In the area of multidimensional databases, the UB-tree represents a promising indexing structure. A key feature of any multidimensional indexing structure is its ability to effectively perform the range queries. In the case of UB-trees, we have proposed an advanced range query algorithm making possible to operate on indices of high dimensionality. In this paper we present experimental results of this range query algorithm.

Keywords: UB-tree, range query, benchmarks, DRU algorithm

1 Introduction

In multidimensional databases, objects are indexed according to several or many independent attributes. However, this task cannot be effectively realized using many standalone indices and thus special indexing structures have been developed is last two decades. Common to all this structures is that they index vectors of values instead of indexing single values.

The UB-tree represents one of the promising multidimensional index structures. Indexing and querying high-dimensional databases is a challenge for current research since high-dimensional indexing is significantly influenced by phenomenon called *curse of dimensionality*. This unpleasant phenomenon states that increasing dimensionality of feature space makes effective indexing and querying very hard (we refer to [2] and [8]). In this paper we present an advanced range query algorithm which makes the UB-tree suitable for indexing large high-dimensional databases.

In Section 1 we describe the UB-tree, Section 2 presents our range query algorithm and Section 3 analyses the comprehensive experimental results. Section 4 concludes the results.

1.1 Universal B-tree

The Universal B-tree (UB-tree) was introduced in [1] for indexing multidimensional data. Its main characteristics reside in an elegant combination of the well-known B^+-tree and the Z-ordering. The power of UB-tree lies in linear ordering of vectors, similarly like an ordering of simple values is indexed by the B^+-tree.

In the UB-tree we require to establish such ordering on a multidimensional vector space and thus linearize the space onto a single-dimensional interval which is usually realized using space filling curves [6]. A space filling curve orders all the points within a n-dimensional vector space. UB-tree was designed to be used with the Z-ordering generated using the Z-curve. Points (tuples) in the space are ordered according to their Z-addresses.

An interval $[\alpha : \beta]$ (α is the lower bound, β is the upper bound) on the Z-curve forms a region in the space which is called *Z-region*. An example of Z-curve and several Z-regions is presented in Figure 1a.

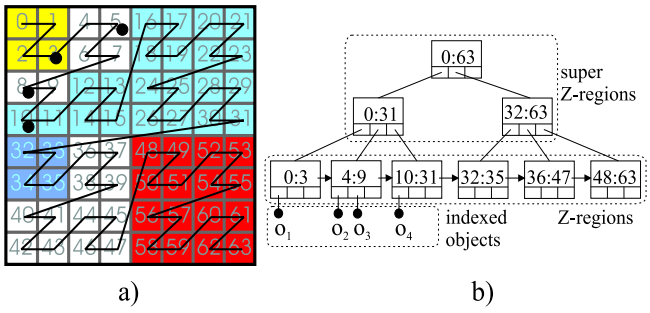


Fig. 1. a) The 2-dimensional space 8×8 filled with the Z-curve. The numbers in the grid are the Z-addresses. The space is partitioned with six Z-regions. The black dots represent 4 indexed objects. b) The UB-tree nodes correspond to the Z-regions and super Z-regions.

Each Z-region is then mapped into a single page within the underlying B^+-tree. The UB-tree leafs represent the Z-regions containing indexed objects themselves while the inner nodes represent the super Z-regions. A *super Z-region* contains all the (super) Z-regions lying entirely inside the super Z-region. Hence, the UB-tree structure is determined by a nested Z-region hierarchy. An indexed vector space and its appropriate UB-tree is depicted in Figure 1.

1.2 Range Queries

Realization of basic operations in the UB-tree (insertion, deletion, point query) is analogous to the operations in the "ordinary" B^+-tree. The main difference is that in the UB-tree we must at first compute the Z-address of the indexed object as a key for the subsequent operation on the underlying B^+-tree.

Unfortunately, a *range query* cannot be so simply forwarded to the B^+-tree. This fact arises from the speciality of the range query which is intended to be used on multidimensional data. Range query (window query respectively) in vector spaces is usually represented with a hyper-box in a given vector space Ω. The ranges of a query box QB are defined by two boundary points, the lower

bound $QB_{low} = [a_1, a_2, \ldots, a_n]$ and the upper bound $QB_{up} = [b_1, b_2, \ldots, b_n]$ where $a_1 \leq b_1, a_2 \leq b_2, \ldots, a_n \leq b_n$. The purpose of a range query is to return all the points located inside the query box, i.e. to return all the points o satisfying $a_i \leq o_i \leq b_i$, for $1 \leq i \leq n$ (see Figure 2a).

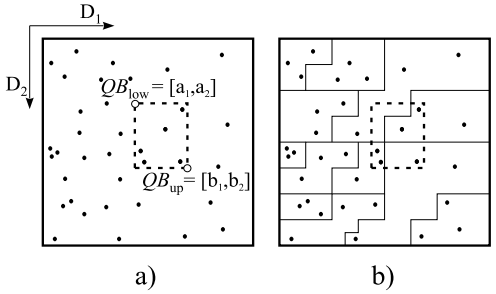


Fig. 2. a) Two-dimensional query box QB specified with lower bound QB_{low} and upper bound QB_{up}. b) Space Ω partitioned to Z-regions.

A more specific range query definition oriented to the UB-tree context can be formulated as a search over all the UB-tree's Z-regions that intersect given query box (see Figure 2b).

Existing Solution. Markl in [4] presents following range query algorithm consecutively searching intersecting Z-regions.

In Figure 3, an example of range query algorithm run is shown. At first, Z-address for the query box lower bound is computed. Using this value a page from UB-tree is retrieved and searched for relevant objects. Next, subsequent Z-region is retrieved and so on. The algorithm will finish as soon as the β of the active Z-region is greater than the Z-address of query box upper bound, i.e. $\beta > Zaddr(QB_{up})$.

So far, the algorithm was elegant and clear. But problem arises when we look deeper into function determining the next Z-address inside the query box. We denote this function `GetNextZaddress`. Computing the next Z-address lying within the query box is not trivial operation since this procedure is obviously dependent on the shape of Z-region. The algorithm for `GetNextZaddress` presented in [4] is of time complexity exponential with the dimensionality. Later, in [5], authors have presented a version linear with the Z-address bit length.

Limitations. Unfortunately, all descriptions of `GetNextZaddress` published so far were mentioned very briefly. Moreover, the explanations were always based on a pure algorithmic basis using "handling with bits" and hence lacking a geometric

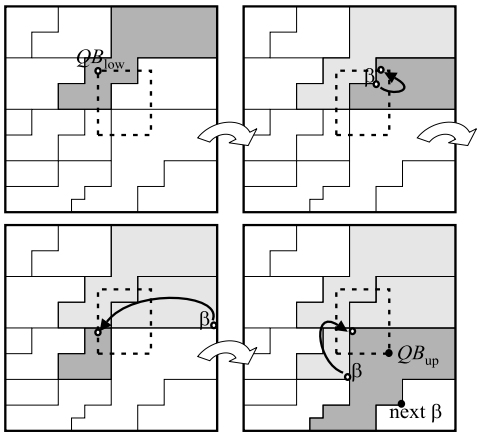


Fig. 3. Range query processing example

model providing a little bit deeper abstract view. Finally, original algorithms on UB-trees are protected with international patents[1].

2 Down-Right-Up Algorithm

The *DRU (Down-Right-Up) algorithm* exploits two types of leaf optimizations reducing unnecessary disk accesses as well as the Z-region intersection computations. The first optimization called *neighbour first point* is used for testing whether the first point of the right neighbour leaf (its Z-region respectively) lies inside the query box. If it does, the algorithm can simply "jump right" (the leafs are linked) to the neighbour leaf and continue processing. This optimization was already used in the original Bayer-Markl's algorithm.

The second optimization called *neighbour region* is specific to the DRU algorithm and is based on existence of `TestZregionIntersection` operation. This operation tests whether a given query box intersects a Z-region. We closely describe an algorithm realizing this operation as well as the theory to DRU algorithm itself in [7].

The *neighbour region* optimization is used for testing whether the neighbour leaf (its Z-region respectively) is intersecting the query box. If it does, the algorithm "jumps right" similarly like by the first optimization.

The DRU algorithm description:
The algorithm uses a *path stack* to keep the actual path in the UB-tree. The path stack allows us to avoid disk accesses to the nodes (and items in nodes) already processed.
DRU algorithm steps (input is the query box *qb*):

[1] Deutsches Patentamt Nr. 197 09 041.9 and Nr. 196 35 429.3

1. Find a leaf the Z-region of which contains Zaddr(qb_{low}). Store the path on the stack.
2. Search actual leaf for tuples lying inside qb. Return these tuples as a part of the result.
3. Retrieve the neighbour leaf from disk and set it as the actual leaf. If the first point of the actual leaf lies inside qb then goto step 2. This is the *neighbour first point* optimization.
4. If the Z-region of the actual leaf intersects qb goto step 2. This is the *neighbour region* optimization.
5. The stack must recover after the "optimization jumps". The UB-tree is passed (along the path in the stack) to the next relevant node. After the recovery, on the top of stack is a parent node of the leaf reached by the preceding optimization.
6. (**Right-Phase**). Peek the node on the top of the stack and try to find a link to the next relevant node (i.e. to node the Z-region of which intersects qb). If no such node is found, pop a node from the stack and repeat step 6 (**Up-Phase**). If a node is found, retrieve the node from the disk and push it onto the stack (**Down-Phase**). If a leaf is reached goto step 2 otherwise repeat step 6.

The algorithm terminates (in any phase) until all relevant pages were examined, i.e. a Z-region was examined such that $\alpha \geq$ Zaddr(qb_{up}).

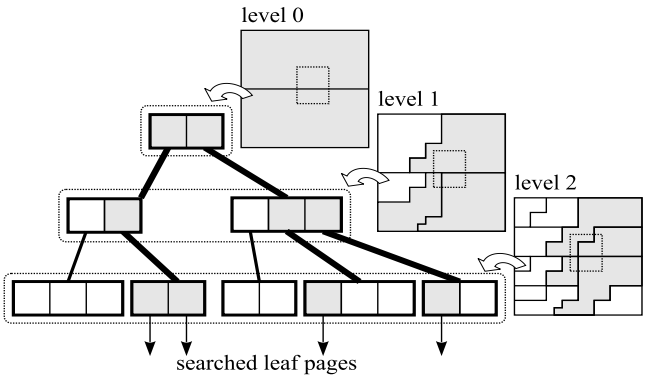


Fig. 4. DRU algorithm

An example of DRU algorithm is presented in Figure 4. Only the intersecting Z-regions (nodes respectively) are being processed. These Z-regions (nodes) are grayed. The bold branches are the only paths passed down. On the leaf level, all tuples lying inside the query box are returned as a query result.

3 Experimental Results

The tests were focused on several cost-factors. Besides the disk access costs (DAC), the effectivity of leaf optimizations was examined. Furthermore, computation costs (CC) of the range queries were investigated.

3.1 Cost Model

Let us now discuss the disk access costs and the computation costs. Let h be the height of the UB-tree, r be the number of Z-regions intersecting the query box, m_f be the number of neigbour leafs matched by the *neighbour first point* optimization, and m_r be the number of neighbour leafs matched by the *neighbour region* optimization.

The basic DAC $= (h + 1) \times r$. Had we consider the *neighbour first point* optimization, the DAC will be reduced to $(h+1)\times(r-m_f)+m_f$. Considering both leaf optimization will reduce the DAC to $(h+1)\times(r-(m_f+m_r))+(m_f+m_r)$.

The asymptotic CC $= \theta((h+2)\times r)$. Had we consider the *neighbour first point* optimization, the asymptotic CC will be reduced to $\theta((h+1)\times(r-m_f)+r)$. Considering both leaf optimizations will reduce the CC to $\theta((h+1)\times(r-m_f)-h\times m_r+r)$.

The set of tests was made on synthetic datasets of increasing dimensionality. The tuples were generated into randomly located clusters of fixed radius (using the L_2 metric) and indexed with the UB-tree. The number of tuples was increasing with the number of dimensions. In order to obtain solid results we have tested large datasets (up to 8 milion 30-dimensional tuples).

Query boxes of various shapes were generated randomly according to the distribution of tuples in the space. The ranges of query boxes were for growing dimensionality the same thus the volumes were increasing but the ratio query box volume/space volume was decreasing. This query box construction is typical for multidimensional applications. The number of queries was increasing with the number of dimensions (from 24 to 120 queries). The results are averaged.

The tests were performed on an Intel Pentium®4 2.4GHz with 512MB DDR333, 60GB UDMA100 7200rpm, run under Windows XP.

3.2 Two-Dimensional Datasets

For the two-dimensional datasets, we have examined the performance dependence on the growing UB-tree node capacity. In general, the greater node capacity implies lower disk access costs and number of computations.

UB-tree characteristics:			
$\|D_i\|$	2^{32}	dimensions	2
tuples	524,288	tree height	4–8
Z-regions	121,138–21,472		
node capacity	6–35	utilization	72.7–69.7%
node size	116–580B	index file	17.4–12.4MB
Range query characteristics:			
range queries	24	query selectivity (tuples)	9676.1
query real times	0.09–0.06 s		

Figure 5a shows the number of leaf Z-regions in two-dimensional UB-tree indices in order to node capacity. We can see that the growing leaf capacity "inflates" the Z-regions' volume while the number of Z-regions decreases. Figure 5b shows the number of Z-regions intersecting the query box. This indicates that the total volume of intersecting Z-regions is not dependent on the growing node capacity. In Figure 6, the disk access costs as well as the number of compu-

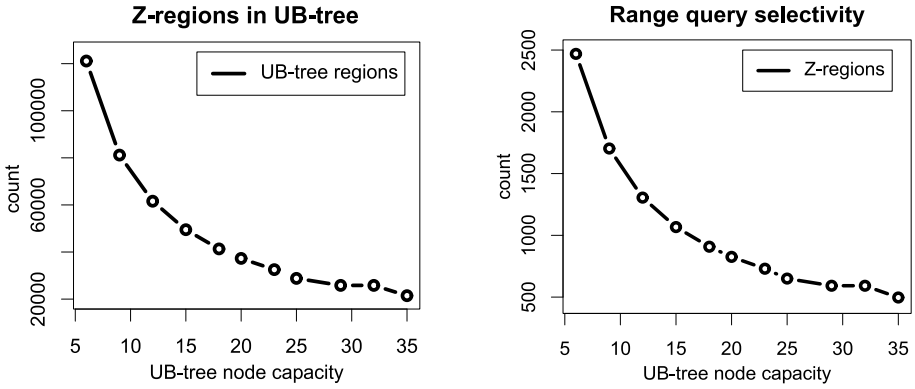


Fig. 5. a) Number of Z-regions. b) Number of Z-regions intersecting the query box.

tations for two-dimensional UB-tree indices are presented. The DRU algorithm performs significantly better than the original Bayer-Markl's algorithm since the DRU algorithms accesses by 30%-60% less disk pages. Similarly, the number of computations is lesser by 25%-50%.

The success of the DRU algorithm is caused by effective application of the *neighbour region* optimization. In Figure 7, the number of matching attempts of the leaf optimizations is presented. Matching attempt means a case when the Z-region *is* intersected, i.e. value *true* is returned and "jump" to the right neighbour leaf is performed. Figure 7a shows that the *neighbour first point* optimization is very effective for lower node capacities. The *neighbour region* optimization is performed after the *neighbour first point* was non-matching. Hence, the result is that for two-dimensional indices (low-dimensional respectively) the *neighbour first point* optimization filters the majority of unnecessary disk accesses. The third line shows the total attempts (matching and non-matching) of either opti-

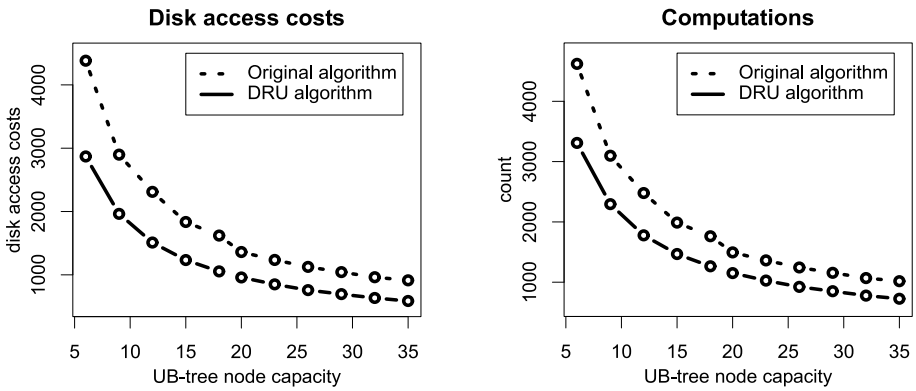


Fig. 6. a) Disk access costs. b) Number of computations.

mization. In Figure 7b, the optimization effectivity is presented. The *neighbour first point* optimization is effective by 90%, together with the *neighbour region* optimization by 95%.

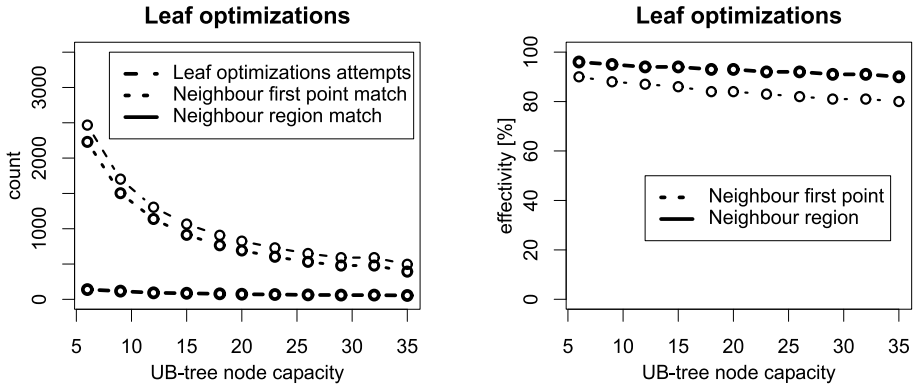


Fig. 7. a) Leaf optimizations. b) Leaf optimizations effectivity.

A particular result is that for low-dimensional UB-trees the Bayer-Markl's algorithm is only slightly less effective than the DRU algorithm. A different situation comes with more dimensions as we will see in the following sub-section.

3.3 High-Dimensional Datasets

In high-dimensional spaces, say for $n \geq 10$, the range query efforts rapidly increase. This fact is caused by the curse of dimensionality described later in this section. In practice, the disk access costs and the number of computations

grow with the increasing dimensionality.

UB-tree characteristics:

$card(D)$	2^{32}	dimensions	2–30
tuples	524,288–7,864,320	tree height	4
nodes	22,400–321,885	Z-regions	21,475–321,885
node capacity	35	utilization	69.7–69.8%
node size	580–4612B	index file	12.4MB–1.44GB

Figure 8a shows the number of inserted tuples according to dimensionality of the dataset. In Figure 8b, the range query selectivity is depicted, i.e. average number of returned tuples. The other line in the graph represents the number of accessed leaf Z-regions. In Figure 9, the disk access costs and the number of

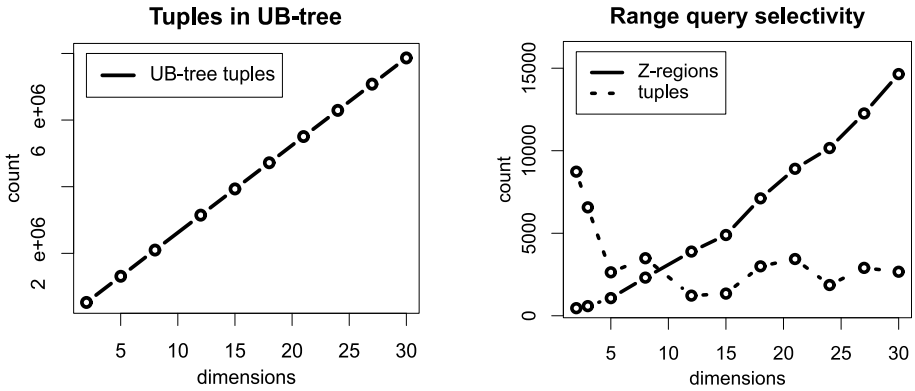


Fig. 8. a) Dataset sizes. b) Range query selectivity.

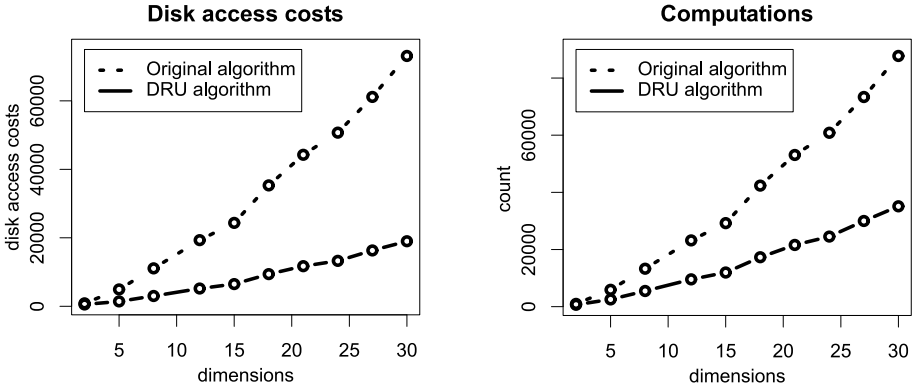


Fig. 9. a) Disk access costs. b) Computations.

computations are presented. We can see that with increasing dimensionality the

costs grow. However, the growth for the DRU algorithm is much less steep than for the Bayer-Markl's algorithm. Thus, the DRU algorithm is more effective for higher dimensionalities. The reason of the DRU algorithm's success resides again in the application of the leaf optimizations.

For higher dimensionalities, the significance of the *neighbour leaf* optimization exponentially grows while the *neighbour first point* optimization goes down to zero. This behaviour is caused by the complex shape of the Z-curve for higher dimensionalities thus the probability that the first point of the neighbour Z-region will intersect the query box tends to zero. On the other side, this kind of probability does not affect the effectivity of the *neighbour region* optimization since it determines the Z-region/query box intersection absolutely.

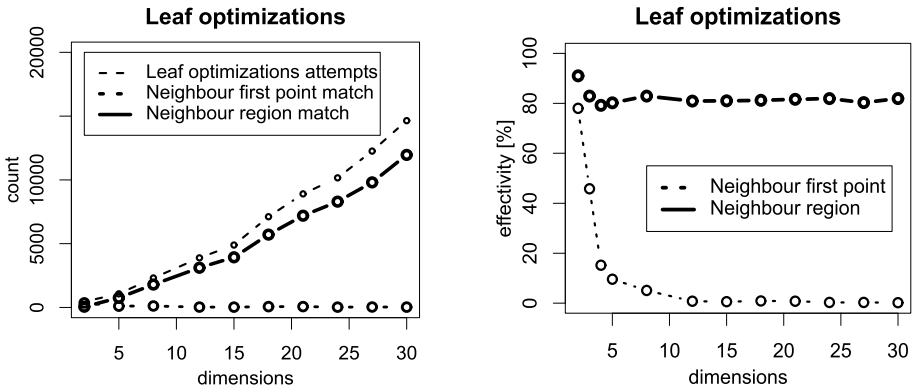


Fig. 10. a) Leaf optimizations. b) Leaf optimizations efficiency.

In Figure 11 we present average real times of a range query execution.

3.4 Curse of Dimensionality

Presented results allow us to think about the *curse of dimensionality* [2], [8] appearing in the UB-tree. With the growing dimensionality of UB-trees grow also the costs, even though less than exponentially. Figure 12a presents a ratio of tuples inside the query box to the number of intersecting Z-regions. Figure 12b shows ratio of intersecting Z-regions containing at least one tuple inside the query box to all of the intersecting Z-regions. This ratio says that in higher dimensionalities more than 95% of relevant Z-regions "give" no tuples to the result. The reason is obvious – the topological properties of the Z-curve are worse for higher dimensionalities.

On the other side, the Figure 12b also shows a ratio of intersecting Z-regions to the Z-regions lying in the interval $[Zaddr(qb_{low}) : Zaddr(qb_{up})]$ (i.e. interval of the query box's "bounding Z-region"). One could expect that the negative effect of the curse of dimensionality will "raise" this ratio up to 100% which is

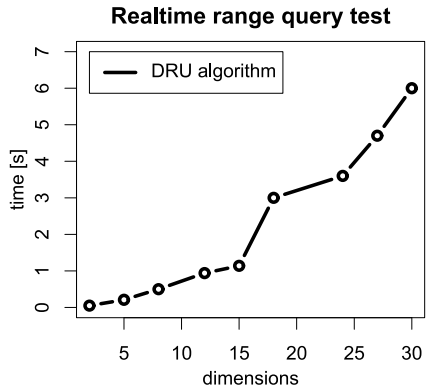


Fig. 11. Range query real times.

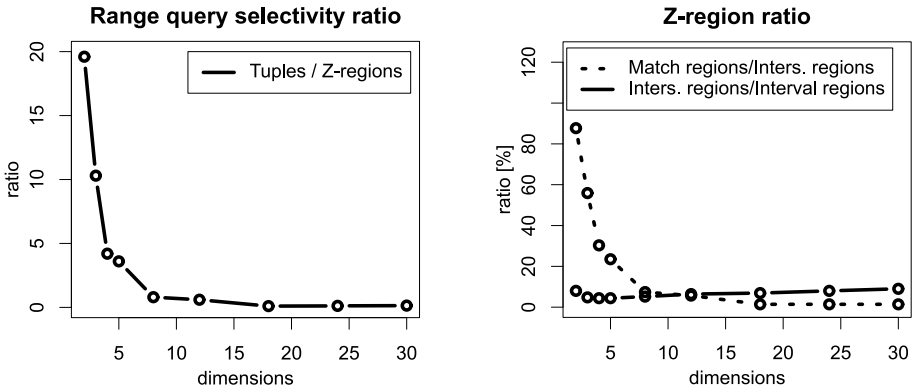


Fig. 12. a) Range query selectivity ratio. b) Query box ratios.

the same as a traversal through the majority of the UB-tree structure. However, this test shows that (even for high dimensionalities) the number of Z-regions intersecting the query box is much lesser than the number of Z-regions within the above mentioned interval. This particular result indicates that the UB-tree together with the DRU algorithm is remarkably resistant to the curse of dimensionality. For a comparison, the well-known *R-tree* [3] used in many applications is very affected by the curse of dimensionality and its usage for high-dimensional indexing is nearly impossible.

4 Conclusions

The experimental results have shown that the DRU range query algorithm makes the UB-tree applicable for effective indexing and querying of high-dimensional feature spaces.

The key to the DRU algorithm effectivity is an incorporation of two leaf optimizations. The *neighbour region* optimization allows the DRU algorithm process range queries in high-dimensional spaces and thus proves that the UB-tree is partially resistable to the unpleasant curse of dimensionality. In particular, the DRU algorithm allows to effectively process "tight" range queries, i.e. query boxes having very disproportionate ranges.

References

1. R. Bayer. The Universal B-Tree for multidimensional indexing: General Concepts. In *Proceedings of World-Wide Computing and its Applications'97, WWCA'97, Tsukuba, Japan*, 1997.
2. C. Böhm, S. Berchtold, and D. Keim. Searching in High-Dimensional Spaces – Index Structures for Improving the Performance of Multimedia Databases. *ACM Computing Surveys*, 33(3):322–373, 2001.
3. A. Guttman. R-Trees: A Dynamic Index Structure for Spatial Searching. In *Proceedings of ACM SIGMOD 1984, Annual Meeting, Boston, USA*, pages 47–57. ACM Press, June 1984.
4. V. Markl, F. Ramsak, and R. Bayer. Improving OLAP Performance by Multidimensional Hierarchical Clustering. In *Proceedings of IDEAS Conference, Montreal, Canada*, 1999.
5. F. Ramsak, V. Markl, R. Fenk, M. Zirkel, K. Elhardt, and R. Bayer. Integrating the UB-tree into a Database System Kernel. In *Proc. Of the 26th Int. Conference VLDB, Cairo, Egypt*, 2000.
6. H. Sagan. *Space-Filling Curves*. Springer–Verlag, 1994.
7. T. Skopal, M. Krátký, V. Snášel, and J. Pokorný. On Range Queries in Universal B-tree. In *submitted to VLDB 2003*, 2003.
8. C. Yu. *High-Dimensional Indexing*. Springer–Verlag, LNCS 2341, 2002.

Clustering Algorithm Via Fuzzy Concepts

Stanislav Krajči

Ústav informatiky, Prírodovedecká fakulta, UPJŠ Košice
krajci@science.upjs.sk

Abstract. The cluster analysis and the formal concept analysis are both used to identify significant groups of similar objects. Rice & Siff's algorithm for the clustering joins these two methods in a case where values of an object-attribute model are 1 or 0 and often reduce an amount of concepts. We define a new type of a fuzzification of a conceptual lattice and we use it for a generalization of this clustering algorithm in a fuzzy case.

1 Introduction

The fields of the cluster analysis and the concept analysis are both used to identify patterns in data. Clusters are groups of objects made using of some distance, concepts are significant groups identified by similarities on their attributes.

Both of these approaches have their disadvantages. There is unclearity in choosing a metric in order the found clusters will be meaningful in some way. On other side, concepts are meaningful but their number may be (and often is) too large to analyze in a reasonable amount of time.

Rice and Siff in [4] make a bridge between the concept analysis and the traditional hierarchical clustering – they define some special (pseudo)metric (or distance) using a classical (crisp) conceptual lattice on a Boolean object-attribute model and then they construct the algorithm for reducing an amount of concepts-clusters using this metric.

We use this approach for the fuzzy case: We define a fuzzification of a conceptual lattice, following the construction of a classical conceptual lattice (as in [1] in the chapter 2. We define some some special metric in the third chapter and we use it for our modified clustering algorithm in the chapter 4. Some characteristics of this algorithm are discussed in the fifth chapter.

2 Fuzzy conceptual lattice

Let an A (like attributes) and a B (like objects) be non-empty (finite) sets and let an R be fuzzy relation on their Cartesian product, i.e. $R : A \times B \to [0, 1]$. This relation can be understood as a table with rows and columns corresponding to objects and attributes respectively. Then the value $R(a, b)$ expresses a grade in which the object b does have the attribute a.

Define a mapping $\tau : \mathcal{P}(\mathcal{B}) \to {}^{A}[0,1]$ which assigns to every set X of elements of B some function $\tau(X)$, a value of which in a point $a \in A$ is

$$\tau(X)(a) = \min\{R(a,b) : b \in X\},$$

i.e. this function assigns to every attribute the least of such values that all objects from X have this attribute at least in such grade.

Conversely, define a mapping $\sigma : {}^{A}[0,1] \to \mathcal{P}(\mathcal{B})$, which assigns to every function $f : A \to [0,1]$ a set

$$\sigma(f) = \{b \in B : (\forall a \in A)R(a,b) \geq f(a)\},$$

i.e. these attributes which have all attributes at least in grade set by the function f (i.e. these attributes the function of their fuzzy-membership to objects dominates over f).

It is easy to see that this mappings have a following properties:

(P1) $X_1 \subseteq X_2 \to \tau(X_1) \geq \tau(X_2)$
(P2) $f_1 \leq f_2 \to \sigma(f_1) \supseteq \sigma(f_2)$
(P3) $X \subseteq \sigma(\tau(X))$
(P4) $f \leq \tau(\sigma(f))$

By $f_1 \leq f_2$ it is understood that for all elements x from a domain (common for both f_1 and f_2) is $f_1(x) \leq f_2(x)$.

These properties are equivalent to the assertion that for each $X \subseteq B$ and $f \in {}^{A}[0,1]$ holds

$$f \leq \tau(X) \text{ iff } X \subseteq \sigma(f)$$

(or, equivalently, the pair $\langle \tau, \sigma \rangle$ is a Galois connection).

Note that in a crisp case, i.e. if the range of the R is (at most) the two-element set $\{0,1\}$, $R(a,b) = 1$ means that the a is an attribute of the object b, and conversely $R(a,b) = 0$ means that the a is not an attribute of the b. Every function from ${}^{A}\{0,1\}$ can be understood equivalently like a subset of the A (because it is its characteristic function). Then $\tau(X)(a) = 1$ iff $R(a,b) = 1$ for all $b \in X$, i.e. $\tau(X)$ is the set of all attributes which all objects from X do have. Conversely it holds $\sigma(\chi_Y) = \{b \in B : (\forall a \in Y)R(a,b) = 1\}$ for every set Y of attributes, hence this is the set of all attributes common to all objects from Y. So our construction is a generalization of the classical Ganter-Wille's one.

Let us continue and define a mapping $\mathrm{cl} : \mathcal{P}(\mathcal{B}) \to \mathcal{P}(\mathcal{B})$ as the composition of the mappings τ and σ: for every $X \subseteq B$ set $\mathrm{cl}(X) = \sigma(\tau(X))$.

It is easy to see from previous properties that cl is a closure operator, i.e. that the following conditions are fulfilled:

(C1) $X \subseteq \mathrm{cl}(X)$
(C2) $X_1 \subseteq X_2 \to \mathrm{cl}(X_1) \subseteq \mathrm{cl}(X_2)$
(C3) $\mathrm{cl}(X) = \mathrm{cl}(\mathrm{cl}(X))$

Like in a crisp case an important role is played by such sets X of objects for which $X = \mathrm{cl}(X)$ (because in such case it holds that if $f = \tau(X)$ then $X = \sigma(f)$). Such pair $\langle X, \tau(X)\rangle$ is called a *fuzzy concept*. Then X is the *extent* of this concept and the corresponding fuzzy set $\tau(X)$ is its *intent*. Because of possibility of reciprocal deriving of both cooordinates of concept it can be considered only one of them, e.g. the first one. Then the set $\mathcal{L}$ of all such extents, i.e. $\mathcal{L} = \{X \in \mathcal{P}(\mathcal{B}) : X = \mathrm{cl}(X)\}$ ordered by inclusion is a lattice, operation of which are defined as following: $X_1 \wedge X_2 = X_1 \cap X_2$ and $X_1 \vee X_2 = \mathrm{cl}(X_1 \cup X_2)$. This lattice we will call a *fuzzy conceptual lattice*.

Note that roles of objects and attributes can be interchanged and this construction works. We have used this converse construction in [3].

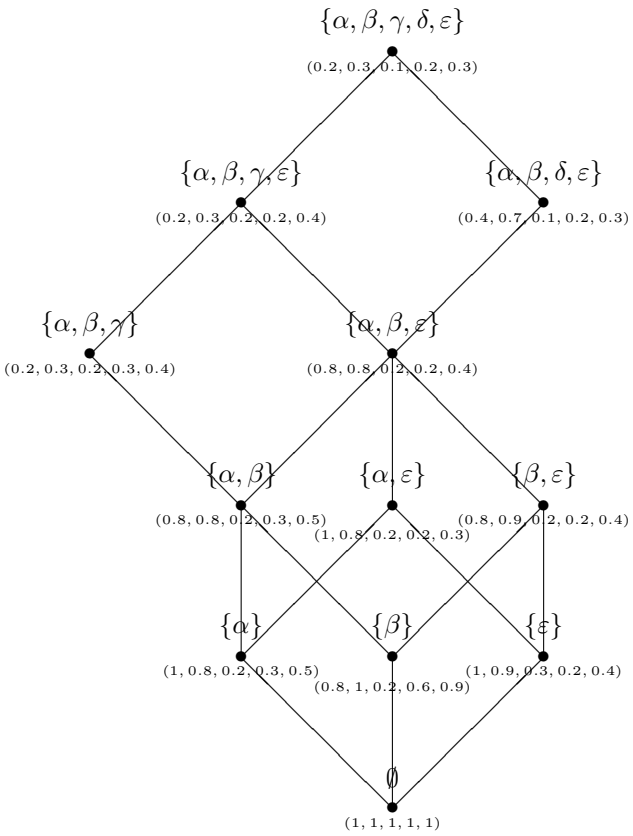
Example

Consider the following fuzzy relation:

	a	b	c	d	e
α	1	0.8	0.2	0.3	0.5
β	0.8	1	0.2	0.6	0.9
γ	0.2	0.3	0.2	0.3	0.4
δ	0.4	0.7	0.1	0.2	0.3
ε	1	0.9	0.3	0.2	0.4

I.e. α, β, γ, δ, and ε are objects and a, b, c, d, e are their attributes. For example, the object α has the attribute a in the grade 1 (i.e. undubiously) and the attribute c in the grade 0.2 (i.e. with big niggles).

The corresponding fuzzy conceptual lattice looks like this:



3 Distance function

Define the function $\rho : \mathcal{P}(\mathcal{B}) \times \mathcal{P}(\mathcal{B}) \to \mathbb{R}$ in the following way:

$$\rho(X_1, X_2) = 1 - \frac{\sum_{a \in A} \min\{\tau(X_1)(a), \tau(X_2)(a)\}}{\sum_{a \in A} \max\{\tau(X_1)(a), \tau(X_2)(a)\}}.$$

It is easy to see that $0 \leq \rho(X_1, X_2) \leq 1$ and the following lemma holds:

Lemma

1. ρ is a pseudometric on $\mathcal{P}(\mathcal{B})$.
2. ρ is a metric on $\mathcal{L}$.

Proof

It is stated in [5] that the vector function $\mathrm{mmm} : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$ defined by

$$\mathrm{mmm}(\langle a_1, \ldots, a_n \rangle, \langle b_1, \ldots, b_n \rangle) = 1 - \frac{\sum_{i=1}^{n} \min\{a_i, b_i\}}{\sum_{i=1}^{n} \max\{a_i, b_i\}}$$

is the (so-called *min/max*) metric on $\mathbb{R}^n$. (Another, an explicit proof can be found in [2].)

Because the function $\tau(X)$ can be understood as a vector (i.e. a row in the table) for every $X \subseteq B$, it holds that $\rho(X_1, X_2) = \mathrm{mmm}(\tau(X_1), \tau(X_2))$ and it implies that ρ is a pseudometric. The prefix "pseudo-" means that there are some $X_1 \neq X_2$ such that $\rho(X_1, X_2) = 0$ or equivalently $\tau(X_1) = \tau(X_2)$). But it can not be the case for elements of $\mathcal{L}$, because then $X_1 = \mathrm{cl}(X_1) = \sigma(\tau(X_1)) = \sigma(\tau(X_2)) = \mathrm{cl}(X_2) = X_2$). It means that the ρ is a metric on $\mathcal{L}$.

Note that this metric is a generalization of Rice and Siff's distance function

$$\rho_{RS}(X_1, X_2) = 1 - \frac{|X_1 \triangle X_2|}{|X_1 \cap X_2|}$$

in the crisp case.

Yet another note: If the w_a is the weight of an attribute a, it can be defined and used the more generalized distance with weights of all attributes:

$$\rho_w(X_1, X_2) = 1 - \frac{\sum_{a \in A} w_a \min\{\tau(X_1)(a), \tau(X_2)(a)\}}{\sum_{a \in A} w_a \max\{\tau(X_1)(a), \tau(X_2)(a)\}}.$$

The same Lemma can be proved (in the same way) for this distance function ρ_w.

4 Clustering algorithm

We take the hierarchical clustering algorithm from [4] but we replace their (pseudo)metric by our ρ. Then our algorithm can be expressed by the following pseudocode:

$\mathcal{C} \leftarrow \mathcal{D} \leftarrow \{\mathrm{cl}(\{b\}) : b \in B\}$
while $(|\mathcal{D}| > 1|)$ do
{
  $m \leftarrow \min\{\rho(X_1, X_2) : X_1, X_2 \in \mathcal{D}, X_1 \neq X_2\}$
  $\mathcal{E} \leftarrow \{\langle X_1, X_2\rangle \in \mathcal{D} \times \mathcal{D} : \rho(X_1, X_2) = m\}$
  $\mathcal{V} \leftarrow \{X \in \mathcal{D} : (\exists Y \in \mathcal{D})\langle X_1, X_2\rangle \in \mathcal{E}\}$
  $\mathcal{N} \leftarrow \{X_1 \vee X_2 : \langle X_1, X_2\rangle \in \mathcal{E}\}$
  $\mathcal{D} \leftarrow (\mathcal{D} \setminus \mathcal{V}) \cup \mathcal{N}$
  $\mathcal{C} \leftarrow \mathcal{C} \cup \mathcal{N}$
}

The variable $\mathcal{D}$ is the set of clusters-concepts in an actual iteration, the $\mathcal{C}$ is the union of all $\mathcal{D}$ till now. The m is the minimal distance of pairs from $\mathcal{D}$. Then the set $\mathcal{E}$ contains "edges", such pairs of clusters from $\mathcal{D}$ which distance is this m and the variable $\mathcal{V}$ contains "verctices", the ends of such edges. Elements of the set $\mathcal{N}$ are new clusters, joins of edges. A new iteration of the $\mathcal{D}$ replaces clusters form $\mathcal{V}$ by their joins from $\mathcal{N}$.

If values of all used variables in the k-th iteration of the algorithm are marked by the index k, we have in our example:

$\mathcal{D}_0 = \{\{\alpha\}, \{\beta\}, \{\alpha, \beta, \gamma\}, \{\alpha, \beta, \delta, \varepsilon\}, \{\varepsilon\}\}$
$\mathcal{C}_0 = \{\{\alpha\}, \{\beta\}, \{\alpha, \beta, \gamma\}, \{\alpha, \beta, \delta, \varepsilon\}, \{\varepsilon\}\}$
$m_1 = \frac{3}{29}$
$\mathcal{E}_1 = \{\langle\{\alpha\}, \{\varepsilon\}\rangle, \langle\{\varepsilon\}, \{\alpha\}\rangle\}$
$\mathcal{V}_1 = \{\{\alpha\}, \{\varepsilon\}\}$
$\mathcal{N}_1 = \{\{\alpha, \varepsilon\}\}$
$\mathcal{D}_1 = \{\{\alpha, \varepsilon\}, \{\beta\}, \{\alpha, \beta, \gamma\}, \{\alpha, \beta, \delta, \varepsilon\}\}$
$\mathcal{C}_1 = \{\{\alpha\}, \{\beta\}, \{\alpha, \beta, \gamma\}, \{\alpha, \beta, \delta, \varepsilon\}, \{\varepsilon\}, \{\alpha, \varepsilon\}\}$
$m_2 = \frac{14}{37}$
$\mathcal{E}_2 = \{\langle\{\alpha, \varepsilon\}, \{\beta\}\rangle, \langle\{\beta\}, \{\alpha, \varepsilon\}\rangle\}$
$\mathcal{V}_2 = \{\{\alpha, \varepsilon\}, \{\beta\}\}$
$\mathcal{N}_2 = \{\{\alpha, \beta, \varepsilon\}\}$
$\mathcal{D}_2 = \{\{\alpha, \beta, \varepsilon\}, \{\alpha, \beta, \gamma\}, \{\alpha, \beta, \delta, \varepsilon\}\}$
$\mathcal{C}_2 = \{\{\alpha\}, \{\beta\}, \{\alpha, \beta, \gamma\}, \{\alpha, \beta, \delta, \varepsilon\}, \{\varepsilon\}, \{\alpha, \varepsilon\}, \{\alpha, \beta, \varepsilon\}\}$
$m_3 = \frac{7}{24}$
$\mathcal{E}_3 = \{\langle\{\alpha, \beta, \varepsilon\}, \{\alpha, \beta, \delta, \varepsilon\}\rangle, \langle\{\alpha, \beta, \delta, \varepsilon\}, \{\alpha, \beta, \varepsilon\}\rangle\}$
$\mathcal{V}_3 = \{\{\alpha, \beta, \varepsilon\}, \{\alpha, \beta, \delta, \varepsilon\}\}$
$\mathcal{N}_3 = \{\{\alpha, \beta, \delta, \varepsilon\}\}$
$\mathcal{D}_3 = \{\{\alpha, \beta, \gamma\}, \{\alpha, \beta, \delta, \varepsilon\}\}$
$\mathcal{C}_3 = \{\{\alpha\}, \{\beta\}, \{\alpha, \beta, \gamma\}, \{\alpha, \beta, \delta, \varepsilon\}, \{\varepsilon\}, \{\alpha, \varepsilon\}, \{\alpha, \beta, \varepsilon\}\} = \mathcal{C}_2$
$m_4 = \frac{7}{20}$
$\mathcal{E}_4 = \{\langle\{\alpha, \beta, \gamma\}, \{\alpha, \beta, \delta, \varepsilon\}\rangle, \langle\{\alpha, \beta, \delta, \varepsilon\}, \{\alpha, \beta, \gamma\}\rangle\}$
$\mathcal{V}_4 = \{\{\alpha, \beta, \gamma\}, \{\alpha, \beta, \delta, \varepsilon\}\}$
$\mathcal{N}_4 = \{\{\alpha, \beta, \delta, \varepsilon\}\}$
$\mathcal{D}_4 = \{\{\alpha, \beta, \gamma, \delta, \varepsilon\}\}$
$\mathcal{C}_4 = \{\{\alpha\}, \{\beta\}, \{\alpha, \beta, \gamma\}, \{\alpha, \beta, \delta, \varepsilon\}, \{\varepsilon\}, \{\alpha, \varepsilon\}, \{\alpha, \beta, \varepsilon\}, \{\alpha, \beta, \gamma, \delta, \varepsilon\}\}$
stop

We have reduced the number of concepts in this way: (extents of) the concepts $\emptyset$, $\{\alpha, \beta\}$, $\{\beta, \varepsilon\}$, and $\{\alpha, \beta, \gamma, \varepsilon\}$ are omitted, because they are not clusters in this meaning.

5 Conclusions

Let us say some words about some characteristics of this algorithm. The finiteness follows from the finiteness of the sets of objects and attributes. If we have the square table with 0's on the diagonal and with 1's elsewhere, then all object play the same role, every subset of them is concept and no of these concepts is removed by our algorithm. It means that its complexity is exponential in general. But this is a rather spiteful example. If the set $\mathcal{E}_k$ has (at most) two elements (i.e. the pair of the minimal distance is only one) in every stage k, the number of clusters is at most the double of the number of objects. In Rice and Siff's crisp case such "exponential blowup has never occured in running the algorithm on any real-world or randomly generated contexts" and in all the number of clusters generated is linear to the number of objects. Because of wider possibilities for

values in the table in the fuzzy case, it is very presumable that are the most of values of distances of pairs of concepts will be different and therefore numbers of iterations will not be very large.

References

1. B. Ganter, R. Wille: Formal Concept Analysis, Mathematical Foundation, Springer Verlag 1999, ISBN 3-540-62771-5
2. S. Krajči: Vektorová min/max metrika, ITAT 2002 Malinô Brdo, Košice 2002
3. S. Krajči: Fuzzifikovaný konceptuálny zväz, submitted to MIS 2003, Josefův Důl
4. M. D. Rice, M. Siff: Clusters, Concepts, and Pseudometrics, preprint, http:// science.slc.edu/˜msiff/papers/mfcsit2000.pdf
5. P. N. Yianilos: Normalized Forms for Two Common Metrics, preprint, http:// www.pnylab.com/pny/

Author Index

Editor:	J. Pokorný, V. Snášel
Department:	Department of Computer Science
Title:	DATESO 2003
Place, year, edition:	Ostrava, 2003, 1st
Page count:	110
Edit:	VŠB-Technical University of Ostrava, Ostrava-Poruba, tř. 17. listopadu 15, 708 33
Print:	Vydavatelství UP Olomouc
Edition:	100

Unsaleable

ISBN 80-248-0330-5