DATESO 2006

proceedings of the 6th annual international workshop

VŠB – Technical University of Ostrava,
Faculty of Electrical Engineering and Computer Science,
Department of Computer Science

ISBN 80-248-1025-5

VŠB–TU Ostrava, FEECS, Department of Computer Science
Czech Technical University in Prague, FEE, Dept. of Computer Science & Eng.
Charles University in Prague, MFF, Department of Software Engineering
Czech Society for Cybernetics and Informatics

Proceedings of the Dateso 2006 Workshop

DATESO 2006

Databases, Texts
Specifications, and Objects

http://www.cs.vsb.cz/dateso/2006/
http://www.ceur-ws.org/Vol-176/

April 26 – 28, 2006
Desná – Černá Říčka

DATESO 2006

Technical editor:
Pavel Moravec, pavel.moravec@vsb.cz

VŠB – Technical University of Ostrava,
Faculty of Electrical Engineering and Computer Science,
Department of Computer Science

Page count:	132
Impression:	200
Edition:	1^{st}
First published:	2006

This proceedings was typeset by PDFLaTeX.
Cover design by Pavel Moravec (pavel.moravec@vsb.cz) and Tomáš Skopal.
Printed and bound in Ostrava, Czech Republic by TiskServis Jiří Pustina.

Published by VŠB – Technical University of Ostrava,
Faculty of Electrical Engineering and Computer Science,
Department of Computer Science

Preface

DATESO 2006, the international workshop on current trends on Databases, Information Retrieval, Algebraic Specification and Object Oriented Programming, was held on April 26 – 28, 2006 in Desná – Černá Říčka. This was the 6^{th} annual workshop organized by VŠB-Technical University Ostrava, Department of Computer Science, FEL ČVUT Praha, Department of Computer Science and Engineering and MFF UK Praha, Department of Software Engineering. The DATESO aims for strengthening the connection between this various areas of informatics. The proceedings of DATESO 2006 are also available at DATESO Web site `http://www.cs.vsb.cz/dateso/2006/` `http://www.ceur-ws.org/Vol-176/`.

The Program Committee selected 12 papers (11 full papers and 1 poster) from 14 submissions, based on two independent reviews.

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 Organizing Committee and Amphora Research Group (ARG, `http://www.cs.vsb.cz/arg/`) for preparation of workshop and its proceedings.

March, 2006 V. Snášel, K. Richta, J. Pokorný (Eds.)

Program Committee

Václav Snášel (chair)	VŠB-Technical University of Ostrava, Ostrava
Karel Richta	Czech Technical University, Prague
Jaroslav Pokorný	Charles University, Prague
Vojtěch Svátek	University of Economics, Prague
Peter Vojtáš	Charles University, Prague

Organizing Committee

Pavel Moravec	VŠB-Technical University of Ostrava
Jan Martinovič	VŠB-Technical University of Ostrava
Yveta Geletičová	VŠB-Technical University of Ostrava

Table of Contents

Full Papers

Posters

Using Object And Object-Oriented Technologies for XML-native Database Systems*

David Toth and Michal Valenta

Dept. of Computer Science and Engineering,
FEE, Czech Technical University
Karlovo náměstí 13, 121 35 Praha 2
Czech Republic
dejvikl@gmail.com, valenta@fel.cvut.cz

Abstract. The aim of this article is to explore and investigate possibilities of reuse already known techniques from object and object-oriented processing for effective processing in XML-native database systems. The article provides explanation of impedance problem, specifically impedance mismatch problem of the middleware, semi-orthogonal persistency methods, and also presents results of experiments with XML:DB API implementation over GOODS object persistent data storage and JAXB technology which were done in January 2006 on Dept. of Computer Science FEE CTU in Prague.

1 Introduction

The main motivation for the work was the participation on project of implementation of an application programming interface designed for work with XML-native database system over a persistent object storage. We used GOODS [3] persistent object storage in the project and tried to implement XML:DB API [2] which claims to be a broadly accepted standard for access to XML-native databases. XML:DB API is supposed to play similar role as ODBC or JDBC APIs in accessing relational databases.

The results of above mentioned project were very controversial. The storage and memory efficiency of implementation was very poor, but the efficiency of application development was amazing. It is possible to design and implement the whole system for storing and processing XML documents and adapt it to match a concrete application requirements in one week.

The main idea of the article is then the attempt to investigate the impedance mismatch problem between object oriented programming language and XML data model. The problem is presented from the point of view of two projects based on very different technologies (both projects are implemented in Java programming language):

1. Implementation of XML:DB API over OO persistent storage GOODS.

* The research was partially supported by the grant GAČR 201/06/0648

2. Implementation of a simple application based on JAXB [4] technology.

The efficiency of processing of XML documents in the first project was tested using INEX[1] XML database collection.

While XML:DB API provides a unified interface to XML databases, JAXB approaches above mentioned impedance mismatch problem in opposite way. Individual parts of XML documents are treated as instances of Java classes in JAXB. It means that the developper (programmer) is limited by given XML schema in his object model of the application.

The rest of the article is organized as follows: the section 2 provides our understanding of XML-native database management systems, section 3 discusses in detail the impedance problem, its consequences, and modifications. Section 4 is dedicated to XML:DB API interface and the cases in which it was found useful. Section 5 presents GOODS project and also discusses the principle of semi-orthogonal persistency which is implemented in it. Section 6 provides brief introduction to JAXB technology. Section 7 presents results of our experiments in the form of tables and graphs. The measurements provide information of efficiency of implementations and also of efficiency of design of applications. The last section summarizes our results and suggests the possibilities of effective employing of discussed technologies.

2 XML Databases

This section claims to specify XML-native database, the circumstances of their appearance, their evolution and future. The discussion is done in order of better understanding of impedance problem consequences.

There is no broadly accepted definition what XML-native databases exactly mean. Using and understanding of this term can therefore very differ by individual authors and communities. For example XML database is defined as a system which simply provides XML API and therefore enables to process XML documents in [6].

We will prefer the terminology which is done for example in [7] in our discussion. Its understanding of XML-native databases are like this:

- database system is a system, which enables a mass data processing - it provides operations store and retrieve. Typically it also provides data consistency, redundancy, multiuser access, and crash recovery mechanisms.
- XML-database system is a database system which provides storing and retrieving of XML documents. Typically a query language like XPath or XQuery are supported.
- XML-native database systems are XML-database systems whose inner data representation is XML-compliant.

Why XML-database systems? Their appearance is a consequence of evolution of internet technologies and also the success of XML standard which was posted in 1998 by W3C consortium and accepted by IETF (Internet Engineering Task Force).

The main reasons of XML success are:

- XML is open international standard
- User can define him/her-self which data and in which structure have to be stored; the data and structural information are stored together in one document

The second of above mentioned reasons directly leads to the idea of semantic web [5]. The [5] is a web portal which provides a complete information of a given area. From this idea arised for example SOAP (Simple Object Access Protocol) standard.

B2B applications seems to be todays top area for XML data format employing. These applications are typical for companies which are doing their business in web environment. XML format is used as data exchange format in heterogeneous environment. The amount of transfered data in XML format all over the internet increases rapidly. Individual companies, due to nature of B2B application and due to very complicated business structure of todays market, have to process many XML documents. This requirement of processing is consequently followed by the requirement of data storing. But if the data models of used storages are different from XML (typically relational or object oriented models) then appears the problem of data transformation. These transformations causes the slow-down of the whole system. Moreover - the storing of XML data in non-native data models (relational, object-oriented) brings problem of efficiency of such kind of mapping. All these kind of problems we will address of impedance problem in this article.

Let us provide a small example for illustration of impedance problem: a company obtain orders in the form of XML documents. These orders are transformed and stored in company's (say relational) database. The company needs to use information from the orders for creating a forms for accounting department. The data are retrieved from relational database, transformed into XML documents and using XSLT transformation (posted by accounting department or company) transformed into acceptable format. It is easy to see that large amount of data following this way of several transformation can easily slow-down the whole system.

The aim of XML-native database system is to eliminate above mentioned impedance problem.

3 Impedance Mismatch Problem

We have explained what we mean by impedance problem in previous section. XML-native database systems seems to be perspective mainly due to their ability to overcome it. Let us now discuss the special case of impedance problem called impedance mismatch problem of the middleware (just called impedance mismatch problem) more in detail in this section.

Impedance mismatch problem is an integral part of every interface between different data models. It is embedded inside a mapping layer between these two models.

Impedance mismatch problem can be observed on several layers of software systems[1]. It appears in the layer of middleware in typical database applications — i.e. it is part of interface between programming language (the language of application) and database model (the model of data storage). Nowadays programming languages use almost object data model[2]. Therefore from the viewpoint of XML-native database systems we have to focus our research to mismatch between the data model of OOPL and XML data models.

Actually there are two approaches to the solution of impedance mismatch problem between OO an XML data models:

1. we use an approach that is independent on a concrete database structure; this approach is represented by unified API, that is very similar to ODBC or JDBC solutions,
2. we have to limit the data model of application language in such a way it match concrete application domain data model.

Both possibilities has its advantages and disadvantages. Let us discuss them generally:

Ad 1: Universal API to data storage guarantees uniformity of each application. It means the development of a new application is faster because developers already know how to communicate with data storage, they do not need to study any new techniques of data access.

On the other hand if the real application data requirements (model) is very different from storage data model, we have to investigate a lot of work in a mapping layer.

Ad 2: In this approach we can design the most suitable data storage interface — such that exactly meets requirements of concrete application. It means the application developer can work standalone — without the need of consultation of a database expert for given storage model.

On the other hand this approach requires to application developer to build a data storage API again and again for each concrete application.

One should conclude that the second approach to solution of impedance mismatch problem is too ineffective to be used in practice. But we will try to show in following sections that also this approach can be very usefull and perspective in some specific circumstances.

The advantage of this approach lies in the fact that we have to know exactly the structure of data we are going to process by the application. Hence our application can become very efficient from the viewpoint of data processing. On the other hand we are losing the potential power which is embedded in concept of self-defining XML [8].

[1] It appears at every transformation layer of system — for example at the layer of network communication, at the layer of human-machine communication etc.

[2] The impedance mismatch problem is also addressed in the theory of programming languages, mainly object-oriented languages. Sometimes it is also referred as **semantic gap**.

4 XML:DB API

XML:DB initiative [2] is interested in database-related aspects of XML. Its products consider mainly XML:DB API and XUpdate. Actually XUpdate is only one existing unofficial standard for changing the content of collection of XML documents (XML database). There is no official standard for these purposes, hence many of real XML database implementations use XUpdate provided by XML:DB initiative.

XML:DB API interface is already implemented by many native and non-native XML database systems. The list of them can be found in [2].

According to previous section XML:DB API represents the universal approach to solution of impedance mismatch problem It is universal solution independent on used implementation platform and application programming language. It is simple and very similar to very known and popular solution for relational databases systems — ODBC and JDBC. It is easy to use for application developers.

But this approach has also its problems — mainly object creation performance problem. It also supports only XPath but not XQuery language.

5 GOODS

GOODS (Generic Object Oriented Database System) [3] is implementation of object-oriented database management system. Generally it is intended for storing of persistent objects. It provides TCP/IP interface and also a library modules for access from Java language. These properties makes GOODS a very interesting platform independent solution for many applications.

GOODS is an open source project founded by Konstantin Knizhnik as his PhD thesis. It supports transactional management, distributed transactions, multiuser access, possibility of user-defined solution of data access conflicts, and many other features that are not relevant in the context of our topic.

The main property of GOODS which seems to have major effect using XML data is its transparency to the application developer. Such transparency is realized by semi-orthogonal persistency mechanism. Let us now to explain this mechanism.

Transparency to the application developer means that developer is not bother by a special methods to call the storage (database) methods. From that point of view he/she works only with objects and does not care if the objects are persistent or not (i.e. transient)[3].

The lot of work had been investigated into transparency mechanism in GOODS implementation. Due to this property the development time of application in GOODS is much more shorter than development the same application using classical database API.

[3] Let us remark that this kind of transparency is in contrast with the requirement to have a uniform independent database API

Typical object database systems like Gemstone/S are using mechanism of orthogonal data persistency. It means each object in application can be classified either as persistent or transient (non-persistent; it means the object and its attributes are lost at the end of application). The method how to distinguish between this two kinds of objects is based on principle of accessibility. There is one object at the beginning, which is classified as persistent root object. Then the object is persistent if it is accessible from the persistent root[4]. Objects which are not accessible from persistent root are transient.

It is easy to see that orthogonal persistency is very comfortable for application programmers. They do not care about methods like store and retrieve. But on the other hand this functionality has to be done on background. It results in a very high overhead of such systems for large amount of data.

GOODS does not employ orthogonal persistency completely but only partially, hence semi-orthogonal persistency. The motivation is decreasing of about mentioned overhead of orthogonal persistency implementation. Semi-orthogonal persistency allows to define persistent only such objects that are inherited from object called Persistant.

Such model of persistency is very similar to the approach which we mentioned in our discussion of impedance mismatch problem and its solution in JAXB (section 6). Once again — it means to reduce application data model (at least the part which should remain persistent) to be fully compliant with storage data model. This analogy and also a very rapid application development phase give as the arguments for using GOODS in our measurements.

6 JAXB

JAXB [4] (Java API for XML Bindings) is part of package JWSDP (Java Web Service Development Pack). The work of JAXB is to generate system of Java classes whose structure is equivalent with XML data which are going to be processed by application. The information about the structure of each possible input XML document is included in XML Schema specification. In other words — we have to know the XML Schema of input data before we can start with application design. Java classes which are generated by JAXB represents the (common) data model for the application and also for data storage. These classes have only limited functionality — they provides put and get methods. The data model classes are maintained by JAXB framework and they are available directly to the application.

JAXB framework resides in system in the form of java archives libraries (.jar files), which are to be included into application. Practical using of JAXB consists of several (standard) steps. We omit here the detail description how to set up and work with JAXB, this information can be found in [9] or [4].

[4] We can also say if there is a path beginning from root persistent object and leading by pointers from one object to another to the given object

Using this approach in the context of XML databases requires that the database is able to provide whole XML document on its interface. This requirement is typically met in all XML databases.

7 Efficiency of discussed technologies — Pilot Applications and Measurements

There was designed and developed system XMLStoreExt which implements XML:DB API core level 0 by specification of [2]. XMLStoreExt uses GOODS as a data storage. It can be really regarded as a XML-native database system from the viewpoint of application developer. Unfortunately it does not support any query language like XQuery or XPath.

Then there was designed and implemented system JAXBStore, which also provides basic database functionality like does XMLStoreExt, but except multi-user access, transactional processing and crash recovery.

System XMLStoreExt was tested using database set INEX [1]. JAXBStore used a datafiles generated by an use case which is published on XQuery Use Case pages — see [10] for details. With regard to systems latency the test data size was used only until 15MB.

The measurement of XMLStoreExt was done in following steps:

- reading a XML file into operational memory,
- storing XML into database (through XML:DB API over GOODS),
- reading XML from database and saving it into another file.

The measurement of JAXBStore consists of following steps:

- unmarshalling of XML document (reading XML into its object representation),
- validation document over XML schema,
- change (update) of data in objects representing the XML tree,
- validation of object tree over XML schema,
- marshalling into XML file.

Here are our measurements included in two simple tables:

Table 1. Measurements in XMLStoreExt

XML data in MB	0,1	0,5	1	2	5	10	15
amount of files	5	12	32	72	133	294	442
storage size in MB	0,43	3,7	6,9	14	32	64	97
time in seconds	3	14	24	46	104	207	322

Here are graphical representation of measurements for both technologies.

Table 2. Measurements in JAXBStore

XML data in MB	0,1	0,5	1	2	5	10	15
amount of files	3	15	30	60	150	300	450
time in seconds	4	6	7	8	14	20	27

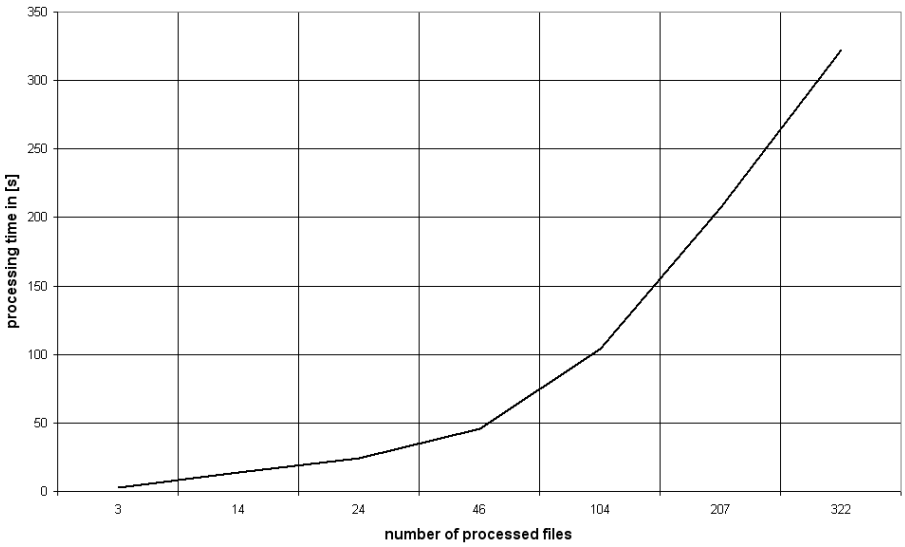

Fig. 1. XMLStoreExt — time dependency on amount of processed files

8 Conclusions

It is easy to see from above mentioned measurements that JAXB technology is faster than GOODS with XML:DB API implementation. On the other hand it is important to remark, that JAXB works only on file level. It does not provide typical database management system features like multiuser access, transactional management, crash recovery and others. All these features are available in GOODS implementation, but with the overhead penalization due to this comfort. GOODS implementation also shows slow-down with the amount of processing data. Also the storage size increases rapidly with the amount of stored data in GOODS implementation. We can see that for 15 MB of input XML data the size of GOODS storage is near to 6times bigger then the input data itself.

Both tested technologies (and applications) shows a tendency to become clogged rapidly with the amount of input data. But they were very good for smaller data amounts. JAXB seems to be a serious candidate for application development, but we have exactly know the structure of data which are going to be processed by application. Measurements also proved our guess that GOODS is very good for very rapid development cycle of application. It seems to be very

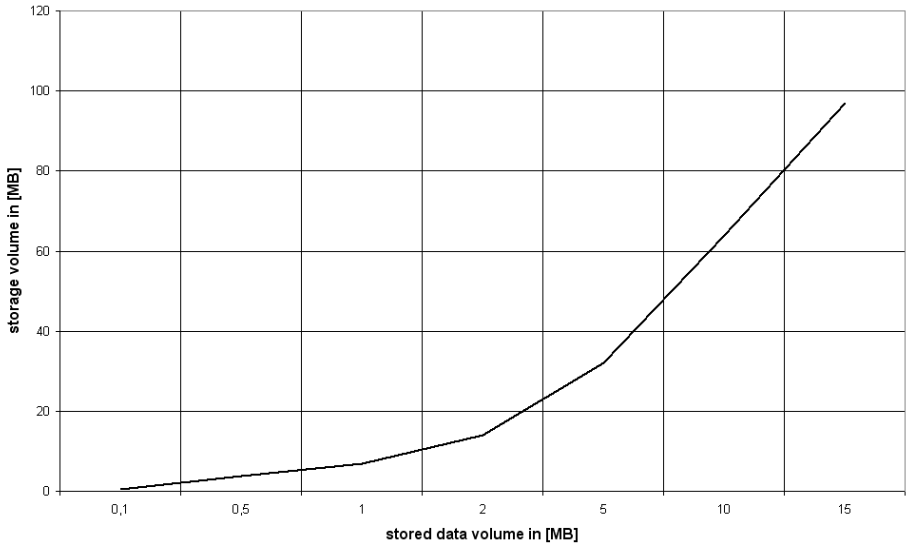

Fig. 2. XMLStoreExt — storage size dependency on size of stored XML data

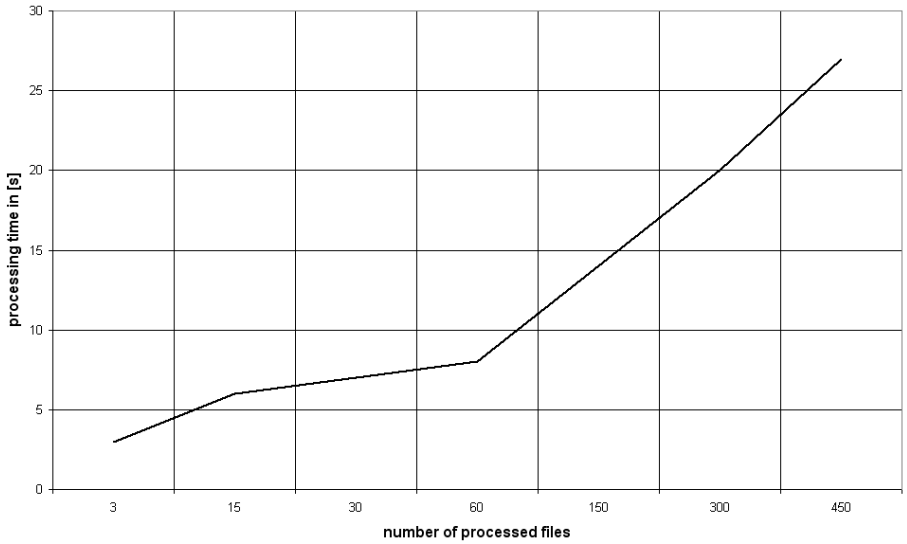

Fig. 3. JAXBStore — time dependency on amount of processed files

suitable for a full functionally prototype application development. It was proved as inefficient for large data.

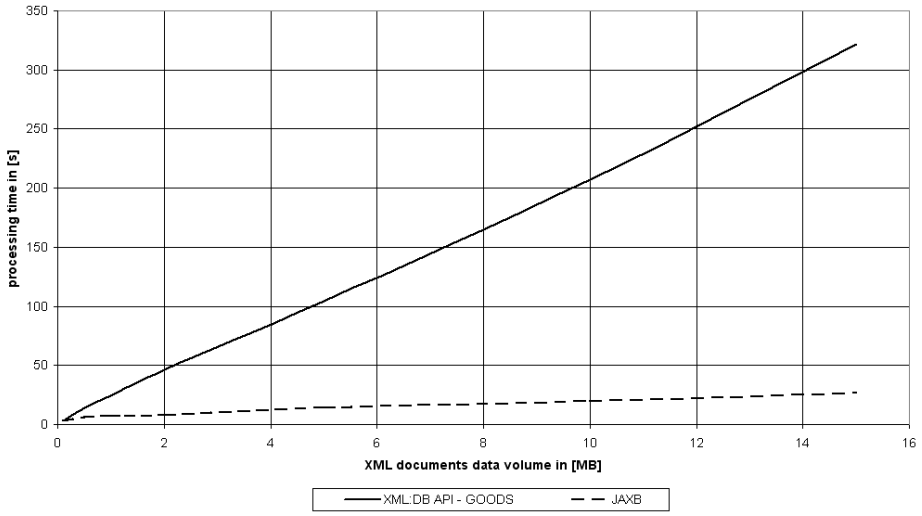

Fig. 4. Efficiency measurement of both referred technologies

We can conclude that application prototyping is suitable domain for both discussed technologies.

References

1. INEX 2003 - home page. http://inex.is.informatik.uni-duisburg.de:2003/index.html.
2. XML:DB initiative - Application Programming Interface for accessing native XML databases. http://xmldb-org.sourceforge.net.
3. GOODS Home Page. http://www.garret.ru/~knizhnik/goods.html.
4. JAXB Home Page. http://java.sun.com/webservices/jaxb/
5. Semantic Web. http://www.semanticweb.org.
6. R. P. Bourret. http://www.rpbourret.com/xml.
7. Chaudri, Rashid, Zicari: XML Data Management. Addision-Wesley. 2003. USA. ISBN 0-201-84452-4.
8. McGoveran: The age of XML databases (self-defining concepts). http://www.eaijournal.com/pdf/XMLMcGoveran.pdf
9. D. Toth: Object And Object-oriented Approaches In XML-native Databases. Master Thesis on Dept. of Computer Science FEE, CTU Prague. Ferbruary 2006 (in Czech).
10. XQuery Use Cases. http://www.w3.org/TR/2005/WD-xquery-use-cases-20050915

Compression of a Dictionary

Jan Lánský and Michal Žemlička

Charles University, Faculty of Mathematics and Physics
Department of Software Engineering
Malostranské nám. 25, 118 00 Praha 1, Czech Republic
zizelevak@gmail.com, michal.zemlicka@mff.cuni.cz

Abstract. Some text compression methods take advantage from using more complex compression units than characters. The synchronization between coder and decoder then can be done by transferring the unit dictionary together with the compressed message. We propose to use a dictionary compression method based on a proper ordering of nodes of the tree-organized dictionary. This reordering allows achieving of better compression ratio. The proposed dictionary compression method has been tested to compress dictionaries for word- and syllable-based compression methods. It seems to be effective for compressing dictionaries of syllables, and promising for larger dictionaries of words.

1 Introduction

Dictionary is used in many applications. Sometimes the space occupied by a dictionary is important and should be taken into account. Then it is reasonable to consider storing the dictionary in a compressed form. We propose here a method for the compression of dictionaries. We have focused on dictionaries used for text compression – or even more precisely: on a compression of dictionaries used by word- or syllable-based document compression methods. The comparisons with other methods are therefore oriented to this topic.

The paper is structured as follows: At first (in part two) we describe the dictionaries and give some formal definitions. Then, in part three, we remember some existing methods used to store the dictionaries. Part four is dedicated to the newly proposed methods. The comparisons of the tested methods are presented in part five. Last part (sixth) is dedicated to the summary.

2 Dictionary

We suppose that a dictionary is a set of ordered pairs (*string*, *number*), where the *string* is a string over an alphabet Σ and the *number* is an integer of the range 1–n where n is the number of the ordered pairs in the dictionary.

It is sometimes useful to partition the set of all *strings* (dictionary items) into several disjoint categories. It is possible that the join of the categories does not cover the set of all possible strings over Σ. In this case it is necessary to ensure that the input strings always fit in the given categories.

For the text compression purposes this requirement can be met e.g. by a proper input string selection (partition of the input message into properly formed subparts). Words and syllables are special types of such strings.

3 Existing Methods for the Compression of a Dictionary

It is quite common for the papers on word- and syllable-based compression methods that their authors give no big importance to the compression of the dictionary as the dictionary often makes only a small part of the resulting compressed message. It is probably true for very large documents but for middle-sized documents the importance of a dictionary size grows as the dictionary takes larger part of the compressed message.

The following two approaches are the most widely used: The first approach is based on coding of a succession of strings (words or syllables) contained in it. In the second approach the dictionary is compressed as a whole. All the strings are concatenated using special separators. The resulting file is then compressed using some general method.

3.1 Compression of Dictionary character-by-character – CD

There is described a method in [1] for the encoding of strings using a partitioning of the strings into five categories, similarly to the method TD3 described below. Every string is encoded as a succession of string type codes followed by encoded string length and by the codes for individual symbols. String type is encoded using binary phase coding (*c1*), string length is encoded by adaptive Huffman code (*c2*), and individual symbols are coded also using adaptive Huffman code (letters by $c3$, numbers by $c4$, and other characters by $c5$). Lower and upper case letters use the same code value $c3$, they are distinguished by the syllable type. All adaptive Huffman trees are initialized according language specification. Examples are given in Fig. 1.

```
code("to") = c1(mixed), c2(2), c3('t'), c3('o')
code("153") = c1(numeric), c2(3), c4('1'), c4('5'), c4('3')
code(". ") = c1(other), c2(2), c5('.'), c5('0')
```

Fig. 1. An example of a coding a string by the CD method

It is not necessary to know the whole dictionary at the beginning. It is possible to compress individual items on the fly. It is then possible to encode new items whenever they are encountered. Other methods discussed in this paper need to compress the whole dictionary at once.

3.2 External Compression of a Dictionary

Let us have a separator being not part of the used alphabet. Let all the strings forming the dictionary are concatenated to a single string using this separator. The resulting string is then encoded using an arbitrary compression method. In [2] the authors tried to encode the dictionary of word using gzip, PPM, and bzip2 methods and recognized as best for this purpose bzip2. We tried to encode the dictionary using bzip2 [3] (in the tables denoted as BzipD – bzip compressed dictionary) and LZW [4] (denoted in the tables as LZWD – LZW compressed dictionary).

4 Trie-Based Compression of a Dictionary

When designing here introduced methods TD1, TD2, and TD3 we decided to represent the dictionary by a data structure *trie* [5, Section 6.3: Digital Searching, pp. 492–512]. Trie T is a tree of maximal degree n, where n is the size of the alphabet of symbols Σ and satisfies following conditions: The root represents an empty element. Let the string α be represented by the node A, the string β represented by the node B. If the node A is father of the node B, then the string β is created by concatenation of the string α by one symbol from Σ. For all nodes A and B exists a node C, that represents common prefix of strings α and β and this node is on both paths (including border points) from the root to B and from the root to A.

The dictionary trie is created from the strings appearing in the text. Then the trie is encoded. Duriung this encoding there is a unique number assigned to each string using depth-first traversal of the trie.

4.1 Basic Version – TD1

Trie compression of a dictionary (TD) is based on coding structure of a trie representing the dictionary. For each node in the trie we know the following: whether the node represents a string (*represents*), the number of sons (*count*), the array of sons (*son*), and the first symbol of an extension for each son (*extension*). Basic version of such encoding (TD1) is given by a recursive procedure *EncodeNode1* in Fig. 2 which traverse the trie by a depth first search (DFS) method. For encoding the whole dictionary we call this procedure on the root of the trie representing the dictionary.

In procedure *EncodeNode1* we code only a number of sons and the distances between the extensions of sons. For non-leaf nodes we must encode in one bit whether that node represents a dictionary item (e.g. syllable or word) or not. Leafs represent dictionary items always, it is not necessary to code it. Differences between extensions of the sons are given as distances of binary values of the extending characters. For coding of a number of sons and the distances between them we use gamma and delta Elias codes [6]. We have tested other Elias codes too, but we achieved the best results for the gamma and delta codes. The numbers of sons and the distances between them can reach the value 0, but standard

```
00  EncodeNode1(node) {
01    output->WriteGamma0(count + 1); /* We encode number of sons */
02    if (count = 0) return;
      /* Mark whether the node represents a string*/
03    if (represents)
04      output->WriteBit(1);
05      else output->WriteBit(0);
06    previous = 0;
      /* We iterate and encode all sons of this modes */
07    for(i = 0; i < count; i++) {
         /* We count and encode distance between sons */
08      distance = son[i]->extension - previous;
09      output->WriteDelta0(distance + 1);
        /* Recursive calling of procedure on the given son */
10      EncodeNode1(son[i]);
11      previous = son[i]->extension;
12    }
13  }
```

Fig. 2. Procedure EncodeNode1

versions of gamma and delta codes starts from 1 what means that these codings do not support this value. We therefore use slight modifications of Elias *gamma* and *delta* codes: $gamma_0(x) = gamma(x+1)$ and $delta_0(x) = delta(x+1)$.

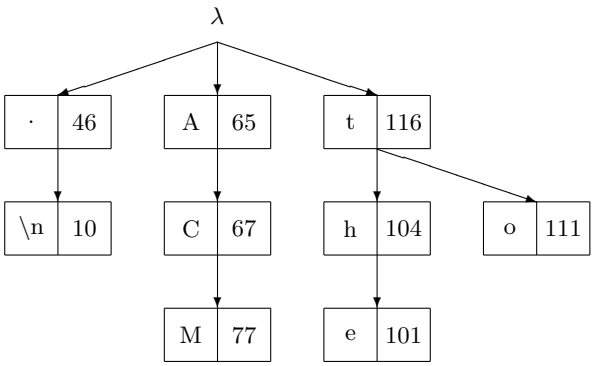

Fig. 3. Example of dictionary for TD1

An example is given in Fig. 3. The example dictionary contains the strings `".\n"`, `"ACM"`, `"AC"`, `"to"`, and `"the"`. Let us introduce the TD1 method by coding the root of the trie representing our example dictionary:

In the node we must first encode the number of its sons. Root has 3 sons, hence we say that $gamma_0$-code of the 3 (sons) is a string of bits '00001' and we write $gamma_0(3) = 00001$.

Then we state that the already represented word (an empty string) is not part of the dictionary by writing a bit 0.

Value of the the first son is encoded as a distance between its value and zero by $delta_0(46 - 0) = 0100101111$.

Then the first subtrie is encoded by a recursive call of the encoding procedure on the first son of the actual node.

When the first subtrie is fully encoded, we should specify what the second son is. The difference between the first and the second son is $65 - 46$, hence we write $delta_0(65 - 46) = 000110011$.

Then we encode the second subtrie and the third son and the subtrie rooted in it. Now the whole node and all it subtries are encoded. As our example node is the root, we have encoded the whole trie representing the dictionary.

4.2 Version with Translator – TD2

In TD1 version the distances between sons according binary values of the extending symbols are coded. These distances are encoded by Elias delta coding representing smaller numbers by shorter codes and larger numbers by longer codes. In version TD2 we reorder the symbols in the alphabet according the types of the symbols and their frequencies typical for given language. In our exmaple the symbols 0–27 are reserved for lower-case letters, 28–53 for upper-case letters, 54–63 for digits and 64–255 for other symbols. There are some examples in table 1.

Table 1. An example of new ordering of the symbols

symbol	'e'	't'	'a'	'I'	'T'	'A'	'0'	'1'	'2'	' '	','	'.'
ord(symbol)	0	1	2	28	29	30	54	55	56	64	65	66

Improving procedure TD1 by a replacement of the expression "$son[i] \rightarrow extension$" by the expression "$ord(son[i] \rightarrow extension)$" in the lines 08 and 11 we get procedure TD2 (Fig. 4).

Let us demonstrate this method on an example (Fig. 5). The example dictionary contains again the strings `".\n"`, `"ACM"`, `"AC"`, `"to"` and `"the"`. We will describe the work of the coding procedure EncodeNode2 on the node labelled by 't'.

In a node we must first encode the number of its sons. Our node has two sons, hence we write $gamma_0(2) = 011$.

Then we state that the already represented word (the string "t") is not part of the dictionary by writing a bit 0.

```
00  EncodeNode2(node) {
01    output->WriteGamma0(count + 1); /* We encode number of sons */
02    if (count = 0) return;
      /* Mark whether the node represents a string*/
03    if (represents)
04      output->WriteBit(1);
05      else output->WriteBit(0);
06    previous = 0;
      /* We iterate and encode all sons of this modes */
07    for(i = 0; i < count; i++) {
         /* We count and encode distance between sons */
08      distance = ord(son[i]->extension) - previous;
09      output->WriteDelta0(distance + 1);
        /* Recursive calling of procedure on the given son */
10      EncodeNode2(son[i]);
11      previous = ord(son[i]->extension);
12    }
13  }
```

Fig. 4. Procedure EncodeNode2

Value of the the first son of `'t'` is encoded as a distance between its value 3 and zero by $delta_0(3-0) = 01100$.

Then the first subtrie of node `'t'` is encoded by a recursive call of the encoding procedure on the first son of the actual node.

When the subtrie of node `'t'` is fully encoded, we should specify what the second son of the root is. The difference between first and second son is $6-3$, hence we write $delta_0(6-3) = 01100$.

Then we encode second subtrie. Now the whole node and all it subtries are encoded.

4.3 Version Using Types of Strings – TD3

Words and syllables are special types of strings. We recognize these five types of words (and syllables): lower-words (from lower-case letters), upper-words (from upper-case letters), mixed-words (having the first letter upper-case and the following letters lower-case), numeric-words (from digits) a other-words (from special characters). We know the type of a coded string for some nodes in the trie (in Fig. 6 *IsKnownTypeOfSons*) and we can use this information.

If a string begins with a lower-case letter (lower-word or lower-syllable), the following letters must be lower-case too. In a trie each son of a lower-case letter can be only a lower-case letter too. Similar situation is for other-words and numeric-words. If a string begins with an upper-case letter, we must look at the second symbol to recognize the type of the string (mixed or upper). In our example (Fig. 5) we know for the nodes `'t'`, `'o'`, `'h'` and `'e'` that all their sons are lower-case letters.

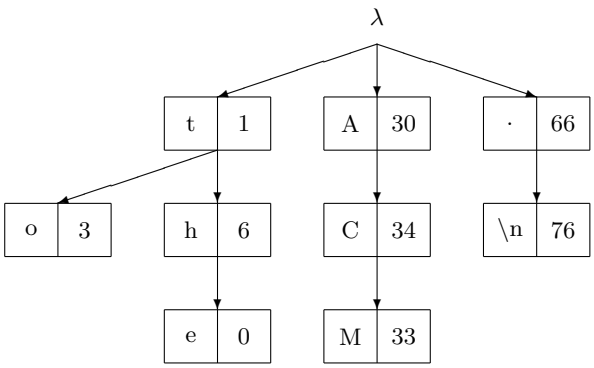

Fig. 5. Example of a dictionary for TD2 and TD3

In the new ordering described in version TD2 it is given for each symbol type some interval of the new orders. Function first returns for each type of symbols the lowest orders available for given symbol type. Function *first* is described in Tab. 2.

Table 2. Values of function first

type of symbols	lower-case letter	upper-case letter	digit	other
first(type)	0	28	54	64

We are counting (Fig. 6, line 10) and coding (Fig. 6, line 11) the distances between the sons. For the first sons of some nodes of a known type, we can use function *first* and decrease the value of the distance and shorten the code. We modify version TD2 by a modifying of the line 06 and inserting the lines 07 and 08 getting version TD3.

Let us show the differences between TD3 and TD2 on our example (Fig. 5).

Let us go directly to the node `'t'`. Here we must first encode the number of the sons of this node (2), we write $gamma_0(2) = 011$.

Then we state that the already represented word (string `"t"`) is not part of the dictionary by writing a bit 0.

Value of the the first son of our node is encoded as a distance between its value (3) and and zero (it is the first son) decreased by value of function $first$ for a lower-case letter (0). Encoded value is $delta_0(3-0-0) = 01100$. It is possible to restrict the shift interval by $first$ of the encoded character type as we know this type – in a subtrie of the node `'t'` occur only lower-case letters. The encoded value is the same as in TD2 but there is a diference is in the calculation.

Other codings are made accordingly.

```
00  EncodeNode2(node) {
01    output->WriteGamma0(count + 1); /* We encode number of sons */
02    if (count = 0) return;
      /* Mark whether the node represents a string*/
03    if (represents)
04      output->WriteBit(1);
05      else output->WriteBit(0);
      /* Using knowledge of type of node (improvent of method TD3) */
06    if (IsKnownTypeOfSons)
07      previous = first(TypeOfSymbol(This->Symbol))
08      else previous = 0;
      /* We iterate and encode all sons of this node */
09    for(i = 0; i < count; i++) {
        /* We count and encode distances between sons */
10      distance = ord(son[i]->extension) - previous;
11      output->WriteDelta0(distance + 1);
        /* Recursive calling of the procedure on the given son */
12      EncodeNode2(son[i]);
13      previous = ord(son[i]->extension);
14    }
15  }
```

Fig. 6. Procedure EncodeNode3

5 Results

We have tested three versions of the method compressing the dictionary using the trie data structure (TD – variants TD1, TD2, TD3), one method compressing the dictionary character-by-character (CD), and two methods using an external compressing tool for the concatenated directory items (LZWD, BzipD).

We have tested the dictionaries of words and syllables for variously sized documents written in following three languages: English (EN), German (GE), and Czech (CZ).

The best for the dictionaries of syllables it appears to be the method TD3 that outperfomed all other tested methods on all tested document sizes. For example, when compressing a 10KB document, TD3-compressed dictionary takes about 770 bytes whereas the second best method (CD) takes about 1450 bytes. In the case of the compression of dictionaries of words the best-performing method has been for small documents (up to 10kB) CD, for middle-sized documents BzipD, and for large documents TD3. The boundary between 'middle-sized' and 'large' documents is in this case dependent on the used language: for Czech it was about 50kB, for English about 200kB and for German about 2MB.

It seems that the success of the TD methods (TD3 inclusive) grows with the average arity of the trie nodes. The syllables are short and the trie representing a dictionary of syllables is typically dense, hence the TD3 method has been always the best.

Table 3. Dictionary of syllables: Compression ratio (Compared with the size of a whole file) in bits per character

—	File size	100 B	1 kB	10 kB	50 kB	200 kB	500 kB	2 MB
Lang.	Method	1 kB	10 kB	50 kB	200 kB	500 kB	2MB	5 MB
CZ	LZWD	5.359	3.233	1.423	0.562	0.343	0.204	—
CZ	CD	3.741	2.432	1.130	0.461	0.284	0.169	—
CZ	BzipD	5.285	2.952	1.227	0.468	0.285	0.168	—
CZ	TD1	4.124	2.232	0.870	0.315	0.185	0.115	—
CZ	TD2	2.944	1.594	0.638	0.240	0.143	0.093	—
CZ	TD3	**2.801**	**1.532**	**0.612**	**0.226**	**0.134**	**0.081**	—
EN	LZWD	4.580	1.715	0.732	0.426	0.269	0.152	0.059
EN	CD	2.983	1.287	0.583	0.360	0.234	0.133	0.052
EN	BzipD	4.390	1.523	0.626	0.353	0.222	0.124	0.047
EN	TD1	3.792	1.276	0.506	0.272	0.158	0.086	0.033
EN	TD2	2.871	0.954	0.384	0.212	0.124	0.069	0.028
EN	TD3	**2.666**	**0.890**	**0.354**	**0.195**	**0.116**	**0.063**	**0.024**
GE	LZWD	4.259	2.995	1.139	0.580	0.345	0.202	0.104
GE	CD	3.068	2.360	0.997	0.530	0.315	0.185	0.091
GE	BzipD	4.127	2.689	0.949	0.479	0.285	0.166	0.087
GE	TD1	3.952	2.539	0.832	0.377	0.207	0.122	0.045
GE	TD2	3.020	1.914	0.627	0.284	0.157	0.097	0.035
GE	TD3	**2.730**	**1.805**	**0.599**	**0.275**	**0.150**	**0.086**	**0.033**

German language has a lot of different and long word forms, the trie representing such dictionary is quite sparse and therefore the TD3 method outperformed other methods only for dictionary of very large documents.

English typically uses less word forms than Czech and German. These word forms are often shorter than the ones used in Czech and German. The trie is then for smaller documents quite sparse and therefore our compression method outperforms the other ones only for larger documents.

In Czech the documents are typically made form lots of middle-sized word forms and the dictionary tries are therefore quite dense. It is the reason why the method has been so successful for the dictionaries of Czech documents.

6 Conclusion

We have proposed three methods for compression of dictionaries based on the representation of the dictionary by a trie data structure. One of them (TD3) has compressed the dictionary of syllables for given files better than all other tested methods have. It has been also the most successful method for compression of dictionaries of words of large documents.

Such dictionaries are used by many word- and syllable-based compression algorithms. Improving compression ratio of the dictionary improves (although with smaller impact) the overall compression ratio of these methods.

Table 4. Dictionary of words: Compression ratio (Compared with the size of a whole file) in bits per character

—	File size	100 B	1 kB	10 kB	50 kB	200 kB	500 kB	2 MB
Lang.	Method	1 kB	10 kB	50 kB	200 kB	500 kB	2MB	5 MB
CZ	LZWD	5.984	4.549	3.076	1.934	1.557	1.161	—
CZ	CD	**4.378**	**3.830**	2.948	1.968	1.648	1.260	—
CZ	BzipD	5.784	4.045	**2.559**	1.582	1.255	0.921	—
CZ	TD1	8.443	6.520	4.146	2.250	1.713	1.178	—
CZ	TD2	5.935	4.531	2.874	1.550	1.176	0.814	—
CZ	TD3	5.781	4.462	2.844	**1.534**	**1.167**	**0.800**	—
EN	LZWD	4.699	2.195	1.203	0.872	0.687	0.443	0.189
EN	CD	**3.100**	**1.776**	1.095	0.847	0.695	0.454	0.197
EN	BzipD	4.508	1.915	**1.002**	**0.714**	0.563	0.361	0.154
EN	TD1	6.320	3.144	1.698	1.108	0.813	0.498	0.191
EN	TD2	4.526	2.142	1.144	0.753	0.554	0.341	0.132
EN	TD3	4.219	2.062	1.110	0.734	**0.544**	**0.333**	**0.128**
GE	LZWD	4.712	3.634	1.819	1.227	0.996	0.706	0.716
GE	CD	**3.582**	**3.091**	1.787	1.293	1.096	0.799	0.789
GE	BzipD	4.409	3.216	**1.506**	**1.001**	**0.797**	**0.558**	0.565
GE	TD1	7.187	5.748	2.585	1.700	1.383	0.945	0.844
GE	TD2	4.985	3.885	1.691	1.094	0.875	0.601	0.534
GE	TD3	4.699	3.776	1.660	1.085	0.867	0.591	**0.532**

References

1. Lánský, J., Žemlička, M.: Text compression: Syllables. In Richta, K., Snášel, V., Pokorný, J., eds.: DATESO 2005, Prague, Czech Technical University (2005) 32–45 Available from http://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS//Vol-129/paper4.pdf.
2. Isal, R.Y.K., Moffat, A.: Parsing strategies for BWT compression. In: Data Compression Conference, Los Alamitos, CA, USA, IEEE CS Press (2001) 429–438
3. Seward, J.: (The bzip2 and libbzip2 official home page) http://sources.redhat.com/bzip2/ as visited on 6th February 2005.
4. Welch, T.A.: A technique for high performance data compression. IEEE Computer **17** (1984) 8–19
5. Knuth, D.: The Art of Computer Programming, Volume 3: Sorting and Searching. Third edn. Addison-Wesley (1997)
6. Elias, P.: Universal codeword sets and representation of the integers. IEEE Trans. on Information Theory **21** (1975) 194–203

Syllable-based Compression for XML Documents

Katsiaryna Chernik, Jan Lánský, and Leo Galamboš

Charles University, Faculty of Mathematics and Physics
Department of Software Engineering
Malostranské nám. 25, 118 00 Praha 1, Czech Republic
kchernik@centrum.cz, zizelevak@gmail.com, leo.galambos@mff.cuni.cz

Abstract. Syllable-based compression achieves sufficiently good results on text documents of a medium size. Since the majority of XML documents are of that size, we suppose that the syllable-based method can give good results on XML documents, especially on documents that have a simple structure (small amount of elements and attributes) and relatively long character data content.
In this paper we propose two syllable-based compression methods for XML documents. The first method, XMLSyl, replaces XML tokens (element tags and attributes) by special codes in input document and then compresses this document using a syllable-based method. The second method, XMillSyl, incorporates syllable-based compression into the existing method for XML compression XMill. XMLSyl and XMillSyl are compared with a non-XML syllable-based method and with other existing method for XML compression.

1 Introduction

The Extensible Markup Language (XML) [5] is a simple text format for structured text documents. XML provides flexibility in storing, processing and exchanging data on the Web. However, due to their verbosity, XML documents are usually larger in size than other exchange formats containing the same data content. One solution to this problem consists of compressing XML documents. Because XML is a text format, it is possible to compress XML documents with existing text compression methods. These methods are more effective, when XML documents have simple structure and long character data content. There are different types of text compression: text compression by characters and text compression by words. There is also a novel method: text compression by symbols [12]. In our work an application of this method to XML documents was developed. Since single text compression is not able to discover and utilize the redundancy in the structure of XML, we modify syllable-based compression method for XML.

At the beginning we supposed that XML syllable-based compression will be suitable for middle-sized textual XML documents. There are many XML documents that meet these conditions, for example any documentation written in DocBook [16] format or news in RSS format [18]. Moreover we suppose that our compression would be more suitable for documents in languages with rich morphology (for example Czech or German [12]).

2 Syllable-based compression method

Syllable-based compression [12] is the method where compression is performed at the syllable level. There are two syllable-based compressors. The first one is syllable-based LZW, and the second one is syllable-based Huffman.

Algorithm LZW [11] is a dictionary compression character-based method. The syllable-based version is called LZWL. In the initialization step, the *syllable dictionary* is filled with empty syllable and syllables from a database of frequent syllables. The following steps are similar with character-based version of LZW, but LZWL works over an alphabet of syllables.

The second syllable-based compression method is called HuffSyllable. It is a statistical compression method based on the adaptive Huffman coding. For our purposes, we use only LZWL syllable-based compression method. Adaptation of HuffSyllable for XML compression gave worse results than LZWL.

3 XMLSyl

Our goal was to modify the syllable-compression method to compress XML documents efficiently. We attempted to modify existing syllable-based compressor so, that it treats XML tokens (element tags and attributes) as single syllables instead of decomposing them into many syllables. There were two possibilities to compel the syllable-based compressor to treat XML tokens as syllables:

1. Modify parser used in the syllable-based tool and combine it with an XML parser, so that it can recognize XML tokens and treat them as a single syllable.
2. Replace XML tokens with bytes in the input document and then compress such a document with an existing syllable-based tool.

We decided to implement the second way because this implementation allows us to make some future improvements easily. For example, we may compel the syllable-based compressor to assign codes with minimal length to XML tokens by adding this single bytes to the *syllable dictionary*[12]. This improvement is impossible in the first variant. The encoding of XML tokens is inspired by existing XML compression methods like XMLPPM [3], XGrind [6], XPress [9], XMill [8].

3.1 Architecture and principles of XMLSyl

The architecture of XMLSyl is shown in Figure 1. It has four major modules: the *SAX Parser*, the *Structure Encoder*, the *Containers* and the *Syllable Compressor*. First, the XML document is sent to the SAX Parser. Next the parser decomposes document into SAX events (start-tags, end-tags, data items, comments and etc.) and forwards them to the Structure Encoder.

The Structure Encoder encodes the SAX events and routes them to the different Containers. There are three containers in our implementation:

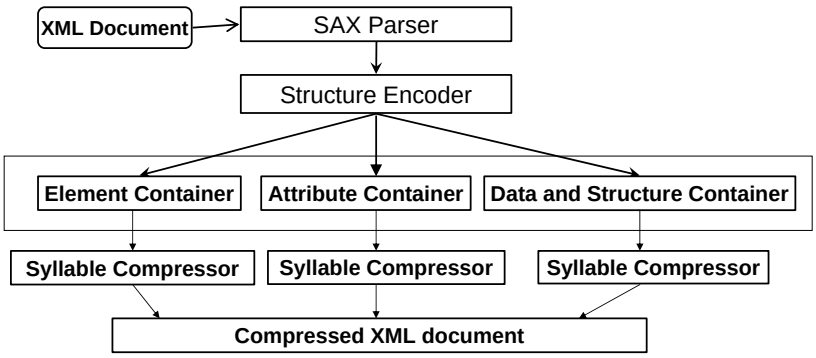

Fig. 1. The Architecture of XMLSylCompressor

1. **Element Container:** The Element Container stores the names of all elements that occur in an XML document. The Structure Encoder also uses the Element Container as the dictionary for encoding XML structure.
2. **Attribute Container:** The Attribute Container stores the names of all attributes which occur in an XML document. The Structure Encoder also uses the Attribute Container as the dictionary for encoding XML structure.
3. **Structure and Data Container:** The Structure and Data Container stores an XML document, in which all meta-data are replaced with special codes. The encoding process is presented in section 3.2.

When a document is parsed and separated into the containers completely, the contents of the containers are sent to the Syllable Compressor. It compresses the content of each container separately using syllable-based compression and sends the result to the output.

We have not written the SAX parser by ourselves, rather we have used the Expat parser[10] which is an open-source SAX parser written in C.

3.2 Encoding the structure of XML document

The structure of XML document is encoded in XMLSyl as follows. Whenever a new element or attribute is encountered, its name is sent to the dictionary and the index of the element is sent to the Data and Structure Container. Two different dictionaries are used for attributes and elements: the Element Dictionary and the Attribute Dictionary. The Attribute Container operates as the Attribute Dictionary and the Element Container as the Element Dictionary. Whenever an end tag is encountered a token `END_TAG` is sent to the Data and Structure container. Whenever a character sequence is encountered, it is sent to the Data and Structure Container without changes. Start and end of character sequences are indicated by special tokens. We distinguish four different character sequences:

value of attribute, value of element, comment, and white spaces between tags, if white spaces are preserved.

To illustrate the encoding process, consider the encoding of the following small XML document:

```
<book>
  <title lang="en">XML</title>
  <author>Brown</author>
  <author>Smith</author>
  <price currency="EURO">49</price>
</book>
<!-- Comment-->
```

First, the XML document is converted into a corresponding stream of SAX events:

```
startElement("book")
startElement("title",("lang","en"))
characters("XML")
endElement("title")
startElement("author")
characters("Smith")
endElement("author")
startElement("author")
characters("Brown")
endElement("author")
startElement("price","currency","EURO")
characters("49")
endElement("price")
endElement("book")
comment("Comment")
```

The tokens in the SAX event stream are sent to the Structure Encoder. It encodes them and sends them to their corresponding containers. When the *book* start element token is encountered, the string *book* is sent to the Element Container since this element name was not encountered before. An index *E0* is assigned to this entry. This index is sent to the Data and Structure Container. The same operation is executed for *title* start element. String *title* is sent to The Element Container and an index *E1* is assigned to it. The index *E1* is sent to the Data and Structure Container. The element *title* has the attribute *lang*. The attribute name is sent to the Attribute Container and the index *A0* is assigned to it. The index *A0* is sent to the Data and Structure Container. Then attribute value *"en"* is sent without modification to the Data and Structure Container. The *"en"* attribute is followed by the token `END_ATT`, that signals the end of the attribute value. When an element value such as *"XML"* is encountered, the token `CHAR`, signaling the beginning of character sequence, the data value and then the token `END_CHAR` are all sent to the Data and Structure Container. Finally, all

the end tags are replaced by the token `END_TAG`. When a comment event is encountered, the code `CMNT` is put into the Data and Structure Container. The comment is also sent to the container and is enclosed by `END_CMNT` code. The final state of all containers is shown in Figure 2.

Element Container

element	index
book	E0
title	E1
author	E2
price	E3

Attribute Container

attribute	index
lang	A0
currency	A1

Data and Structure Container

<book>	<title	lang="en">	XML	</title>	<author>
E0	E1	A0 en END_ATT	CHAR XML END_CHAR	END_TAG	E2

Brown	</author>	<author>	Smith	</author>	<price
CHAR Brown END_CHAR	END_TAG	E2	CHAR Smith END_CHAR	END_TAG	E3

currency="EURO">	49	</price>	</book>	<!--Comment-->
A1 Euro END_ATT	CHAR 49 END_CHAR	END_TAG	END_TAG	CMNT Comment END_CMNT

Fig. 2. Content of containers

In this example we have ignored white spaces between tags, e.g. `<book>` and `<title>`, so the decompressor then produces a standard indentation. Optionally, XMLSyl can preserve the white spaces. In that case, it stores the white spaces as the sequence of characters in the Data and Structure Container between tokens `WS` and `END_WS`.

3.3 Containers

The containers are the basic units for grouping XML data. The Attribute Container holds attribute names and the Element Container holds element names. As long as the number of all element and attribute names in any XML document is not high, this two containers are kept in main memory. During parsing, the containers size increases as the container is filled with entries. Each entry in the Element container is assigned a byte in the range 00-A9. These bytes are used for encoding the element names. Each entry in the Attribute container is assigned a byte in the range AA-F9. These bytes are used for encoding the attribute names. The residual 6 bytes are reserved for special codes like `CHAR`, `END_TAG` etc. In most cases, 170 (or 80) bytes are enough to encode element (or attribute) names. If the number of elements (or attributes) are greater than 170 (or 80), entries are encoded with two bytes, then tree and so on.

There is another situation with The Data and Structure Container. We do not know the size of the input XML document. The size of XML document can be so big, that document will not fit into memory, and it is not possible to increase

the size of container endlessly. Therefore, the container consists of two memory block of constant size. The content of the first memory block is compressed, as soon as the container is filled. We don't compress two blocks at once, because the context of the second memory block is used for compression of the first one. After the compression, the compressed content of the first block is sent to the output and the first block swaps its purpose with the second one. Now the first block is filled with data. When it is full, the second block is compressed, and so on.

3.4 The Syllable Compressor

The Syllable Compressor compresses the Structure and Data Container first and sends the output to the output file. Then the Attribute Containers are compressed and sent to the output file and finally the same happens with the Element Container. LZWL is used for the compression of data. HuffSyll could be also chosen, but the performance is worse, so we decided to use only LZWL.

4 XMillSyl

This chapter introduces our second syllable-based XML compressor, XMillSyl. This second method incorporates syllable-based compression with the existing method for XML compression of XMill [8]. XMill has two main principles in order to optimize XML compression:

- separating structure from data content, and
- grouping Data values with related semantics in the same "container".

Each data container is then compressed individually with gzip [21]. In XMillSyl, containers are compressed with LZWL.

We do not suppose that XMillSyl method gives better results than XMill because gzip compression performs better than LZWL. We have implemented XMillSyl in order to compare the power of XMLSyl with the power of two main principles of XMill.

4.1 Implementation

We did not write XMill compressor. We decided to use existing sources of XMill.

XMill operates as follows: a SAX parser parses the XML file and the SAX events are sent to the core module of the XMill called the path processor. It determines how to map tokens to containers: element tag names and attribute names are encoded and sent to the structure container, while the data values are sent to various data containers, according to their semantic. Finally, the containers are gzipped independently and stored on disk.

We have modified compression and decompression functions (operating on containers) in the way they compress and decompress the data containers with

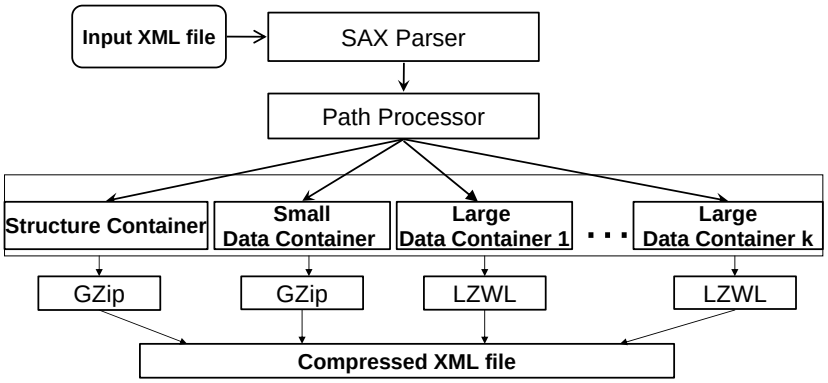

Fig. 3. Architecture of XMillSyl

the syllable-based method (see Figure 3). Moreover we have modified the syllable-based method so that it can work with the containers of XMill implementation instead of a file stream.

XMillSyl discerns the difference between small and large containers. Since LZWL is not suitable for extremely small data, the small containers are compressed with gzip. The structure container is also gzipped in XMillSyl. The large containers are compressed with LZWL.

Table 1. The first data set.

	Size	Lang	Description
elts	103919	English	Periodic table of the elements in XML
pcc	2600257	English	Formal proofs transformed to XML
stats	869059	English	One year statistics if baseball players
tal	1364576	English	Safe-annotated assembly language converted to XML
tpc	313193	English	The XML representation of the TPC_D benchmark database.

5 Comparison Experiments

To show the effectiveness of XMLSyll and XMillSyl, we compared the performance of this two compressors with one representative of XML compressors XMill and the syllable-based compressor LZWL [12].

5.1 XML data sources

XMLSyl and XMillSyl were tested on two data sets that cover a wide range of XML data formats and structures. The first data set is shown in Table 1. It contains English XML documents with different inner structure. It includes regular data that has regular markup and short character data content (elts, stats, weblog, tpc). It also includes irregular data, that has irregular markup (pcc, tall).

The second data set is shown in Table 2. It contains textual XML documents of simple structure with long character data content. It contains five stage plays marked up as XML, four in English and one in Czech. It also contains data in DocBook format in Czech and in English.

Some data was distributed with the XMLPPM [3] and the Exalt [4] compressors while others were found on Internet [15], [16]. All Czech documents use Windows-1250 encoding.

Table 2. The second data set.

	Size	Lan	Description
errors	153530	English	"The Comedy of Errors" marked up as XML
hamlet	314677	English	"The Tragedy of Hamlet, Prince of Denmark" marked up as XML
antony	289865	English	"The Tragedy of Antony and Cleopatra" marked up as XML
much_ado	220495	English	"Much Ado about Nothing" marked up as XML
ch00	13916	English	"DocBook: The Definitive Guide" in DocBook format (1)
ch01	55015	English	"DocBook: The Definitive Guide" in DocBook format (2)
ch02	160728	English	"DocBook: The Definitive Guide" in DocBook format (3)
ch03	27799	English	"DocBook: The Definitive Guide" in DocBook format (4)
ch04	137440	English	"DocBook: The Definitive Guide" in DocBook format (6)
ch05	67142	English	"DocBook: The Definitive Guide" in DocBook format (7)
glossary	24701	English	"DocBook: The Definitive Guide" in DocBook format (8)
howto	42853	English	"DocBook V5.0, Transition Guide" in DocBook format.
hledani	16429	Czech	"Inteligentní podpora navigace na WWW s využitím XML" in DocBook (1)
komunikace	50881	Czech	"Inteligentní podpora navigace na WWW s využitím XML" in DocBook (2)
navihace	18495	Czech	"Inteligentní podpora navigace na WWW s využitím XML" in DocBook (3)
robot	25405	Czech	"Inteligentní podpora navigace na WWW s využitím XML" in DocBook (4)
xml	28467	Czech	"Inteligentní podpora navigace na WWW s využitím XML" in DocBook (5)
rur1	59609	Czech	"R.U.R" marked up as XML.

5.2 Compression Performance Metrics

The compression ratio is defined as follows:

$$CR = \frac{sizeof(compressed\ file) \times 8}{sizeof(original\ file)}\ bits/byte.$$

We compare XMillSyl and XMLSyl compression ratios with those of XMill. The compression ratio factor shows normalization of the compression ratio of

Table 3. The first data set.

	CR_{LZWL}	CR_{Xmill}	$CR_{XMillSyl}$	$CRF_{XMillSyll}$	CR_{XMLSyl}	CRF_{XMLSyl}
elts	1,04	0,47	0,54	**1,15**	0,72	**1,53**
pcc	0,22	0,02	0,03	**1,50**	0,04	**2,00**
stats	0,67	0,33	0,40	**1,21**	0,39	**1,18**
tal	0,36	0,09	0,12	**1,33**	0,15	**1,67**
tpc	1,82	1,05	1,54	**1,47**	1,60	**1,52**
Average	0,82	0,39	0,53	**1,33**	0,58	**1,58**

XMillSyll or XMLSyl with respect to XMill. The compression ratio factor is defined as follows:

$$CRF_{XSyl} = \frac{CR_{XSyl}}{CR_{XMill}}.$$

5.3 Experimental Results

The compression ratio statistics of two sets of XML documents are shown in Table 3 and Table 4.

The syllable-based method performed worse on documents from the first data set. On the other hand, both XMLSyl and XMillSyl shows great improvement comparing to LZWL. They compressed the input to 50-60% of the size of the compressed file with LZWL.

On XML documents of the second data set, LZWL provides a reasonably good compression ratio - on the average, about two-thirds that of XMill. This confirms our prediction, that syllable-based compression is effective for textual XML documents. Moreover our compression methods show even greater improvement.

On the document of the second data set, XMillSyl achieves about 15% and XMLSyl is about 20% better compression ratio than LZWL. Compared to XMill, both methods perform slightly worse. XMillSyl compresses about 13% and XMLSyl about 7% worse than XMill.

Figure 4 shows the variation of the compression ratio as a function of XML data size for "DocBook: The Definitive Guide".The compression was run on several subsets. On small files XMillSyl performs better than XMLSyl. The explanation is, that the data are split into many small containers in XMillSyl, which are compressed with gzip (gzip outperforms LZWL, especially on small data). On middle-sized and large files XMLSyl outperforms XMillSyl. We can observe that the bigger size also implies a better compression.

6 Conclusion

In this work we introduced syllable-based compression tools for XML documents called XMLSyl and XMillSyl. We presented the architecture and implementation

Table 4. The first data set.

	CR_{LZWL}	CR_{Xmill}	$CR_{XMillSyl}$	$CRF_{XMillSyll}$	CR_{XMLSyl}	CRF_{XMLSyl}
errors	1,98	1,83	2,00	**1,09**	1,83	**1,00**
hamlet	1,96	1,91	2,00	**1,05**	1,85	**0,97**
antony	1,84	1,79	1,88	**1,05**	1,69	**0,94**
much_ado	1,88	1,80	1,89	**1,05**	1,77	**0,98**
ch00	3,28	2,69	3,00	**1,12**	2,88	**1,07**
ch01	2,69	2,20	2,43	**1,10**	2,46	**1,12**
ch02	1,76	1,43	1,70	**1,19**	1,57	**1,10**
ch03	2,90	1,87	2,70	**1,44**	2,08	**1,11**
ch04	2,09	1,66	1,78	**1,07**	1,83	**1,10**
ch05	2,28	1,81	2,03	**1,12**	2,04	**1,13**
glossary	2,07	1,64	1,84	**1,12**	1,89	**1,15**
howto	6,69	2,30	2,50	**1,09**	2,59	**1,13**
hledani	3,79	3,13	3,62	**1,16**	3,40	**1,09**
komunikace	3,25	2,65	2,93	**1,11**	3,01	**1,14**
navihace	3,79	3,14	3,68	**1,17**	3,44	**1,10**
robot	3,43	2,86	3,22	**1,13**	3,04	**1,06**
xml	3,74	3,23	3,69	**1,14**	3,30	**1,02**
rur1	2,33	2,07	2,37	**1,14**	2,15	**1,04**
Average	**2,88**	**2,22**	**2,51**	**1,13**	**2,38**	**1,07**

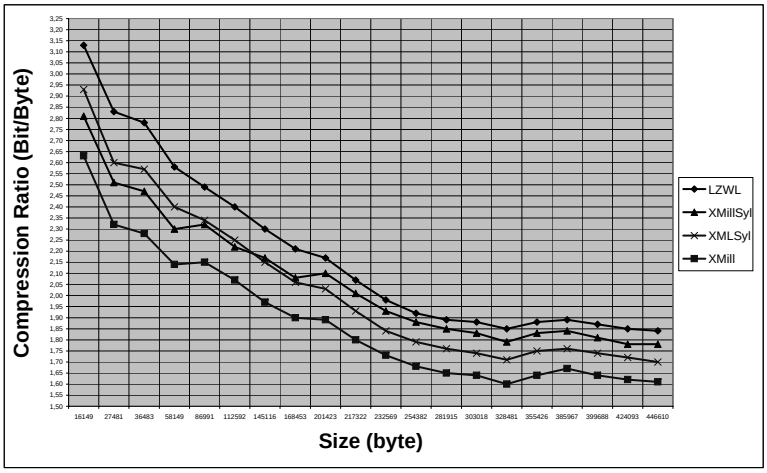

Fig. 4. Compression ratio under different sizes.

of our tools and tested their performance on a variety of XML documents. In our experiments, XMLSyl and XMillSyl were compared with LZWL and XMill. Both methods are more suitable for textual XML documents. XMill outperformed our methods only marginally. XMLSyl performs better than XMillSyl. It implies that in our case encoding of XML structure is more efficient than separating a structure from data and grouping data values with related meaning. XMillSyl and XMLSyl show better results for Czech language.

In the future, we want implement some modifications to enhance the compression ratio. For example, the information in the DTD section can be extracted and utilized to create a special syllable dictionary for elements and attributes.

References

1. Wilfred Ng, Lam Wai, Yeung James Cheng. Comparative Analysis of XML Compression Technologies. World Wide Web Journal, 2005
2. Smitha S. Nair. XML Compression Techniques: A Survey. `www.cs.uiowa.edu/~rlawrenc/research/Students/SN_04_XMLCompress.pdf`
3. J. Cheney. Compressing XML with Multiplexed Hierarchical PPM Models *In Proc. Data Compression Conference*, 2001.
4. V. Toman. Compression of XML Data. MFF UK, 2003
5. World Wide Web Consorcium. Extensive Markup Language (XML) 1.0. `http://www.w3.org/XML/`
6. P. Tolani, J. R. Haritsa. XGrind: A Query-friendly XML Compressor. *In Proc. IEEE International Conference on Data Engineering*, 2002.
7. SAX: A Simple API for XML. `http://www.saxproject.org`
8. H. Liefke, D. Suciu. XMill: an Efficient Compressor for XML Data. *In Proc. ACM SIGMOD Conference, 2000.*
9. *Jun-Ki Min, Myung-Jae Park, Chin-Wan Chung, XPRESS: A Queriable Compression for XML Data* SIGMOD 2003, June 912, 2003, San Diego, CA, 2000.
10. Expat XML Parser. `http://expat.sourceforge.net`
11. T. A. Welch. A technique for high performance data compression. *IEEE Computer, 1984.*
12. *J. Lansky, M. Zemlicka. Text Compression: Syllables.* DATESO, 2005
13. J. Lansky, Slabiková komprese. MFF UK, 2005
14. V. Toman. Komprese XML dat. `http://kocour.ms.mff.cuni.cz/~mlynkova/prg036/`
15. J. Kosek. Inteligentní podpora navigace na WWW s využitím XML. `http://www.kosek.cz/diplomka/`, 2002
16. DocBook `http://www.docbook.org/`
17. A Quick Introduction to XML. `http://www.cellml.org/tutorial/xml_guide`
18. M. Pilgrim. What Is RSS. `http://www.xml.com/pub/a/2002/12/18/dive-into-xml.html`
19. XML Processing. `http://diveintopython.org/xml_processing/`
20. SAX And DOM Overview. `http://www.jezuk.co.uk/cgi-bin/view/arabica/SAXandDOMIntro`
21. The gzip home page. `http://www.gzip.org/`

CellStore – the Vision of Pure Object Database

Jan Vraný

Department of Computer Science, FEE, Czech Technical University in Prague,
Karlovo náměstí 13, 120 00, Praha, Czech Republic
vranyj1@fel.cvut.cz

Abstract. This paper describes a vision of CellStore, a kind of universal database system, which would be capable of storing and operating on several different data models – object, network, hierarchical and even relational one. Features of CellStore will be described as well as underlying storage model and database architecture.

1 Motivation

The world's mainstream programming paradigm for robust, large scale, mission-critical application is object-oriented programming (OOP). Many of such applications need support of database to maintain its data. But nobody doubts that the database should be relational or object-relational one. The semantic gap between those two totally different paradigms brings some problems, that has to be solved. Basically, there are three possible solutions:

- The application operates on data in a "relational way", i.e. the programmer has to use SQL queries to access data directly. In this case, usage of objects is limited only to usage of OO libraries for GUI and so on.
- Some kind of object-relational mapper is used (GLORP [5] or Hibernate [6] are examples of such O-R mappers). This allows programmers to manipulate data in a "object" way, but architecture and capabilities of O-R mapper limits the design of application and underlying database schema.
- Network or object database is used instead of relational one.

2 Currently available object-oriented databases

There are currently many so-called "object databases" – OmniBase, DB4Objects, ZODB, GOODS, Elephant, GemStone/S. In fact many of them are network rather than object ones. Both network and object database are very similar. Both can store any arbitrary object structure. The difference is that an object database also stores code (methods) together with regular data. Object database can execute any code stored in it itself, no client environment is needed.

2.1 OmniBase

OmniBase [7] is embedded network database written in smalltalk. It is available for many different smalltalk dialects – Doplhin Smalltalk, Squeak, VisualWorks, Smalltalk/X and VAST. OmniBase supports multi-version concurrency control, object clustering, online backups and thread-safe operations.

Garbage collecting is supported, but cannot be performed on live database.

2.2 DB4Objects

DB4Objects [8] is less or more similar to OmniBase, but there are two differences:

1. DB4Objects is targeted on Java and .NET (C#) platforms.
2. DB4Objects can operate as embedded database or can run as normal database server which communicates with clients over the network.

2.3 GemStone/S

GemStone/S [9] is full-featured object database based on smalltalk dialect called Smalltalk DB. GemStone application consists of three parts: a client (usually VisualWorks smalltalk), a Gem (a part of GemStone responsible for evaluating, transaction processing and so on) and a Stone (a part responsible for managing low-level storage). Each part can run on different node in a network. A special Gem called GcGem is responsible for garbage collecting, which is performed during normal processing of client requests.

3 The CellStore project

The basic motivation for CellStore project was development of an experimental database, which can be used as a basis for experimenting with various database algorithms like locking and caching strategies, transaction policies, different data type models, etc.

The project is divided into three relatively independent parts:

- *CellStore low-level storage*, which provides a basic storage management,
- *CellStore/OODB*, an experimental object database,
- *CellStore/XML*, an experimental native XML database.

The design and implementation of CellStore is focused more on simple OO design and modularity than on implementation performance. As long as CellStore is experimental system, saving several bytes of memory or several processor instructions doesn't matter.

3.1 The low-level storage model

The low-level storage model gave CellStore its name. It is combination of storage models of Lisp, Smalltalk and Oracle RDBM. Basically, the storage is divided into two main spaces.

- *Cell space*, which contains only the structural information about stored data. Structure is kept in fixed-size *cells*. Each cell has several *fields*, which can contain pointer to another cell in the cell space or a pointer to a record in the data space. Cells describe only relationships between data elements (objects) stored in a CellStore database.
- *Data space*, which contain actual data, i.e. a byte arrays. Data space is organised into blocks, each block may contain several *records*. Each record in data space is identified by a unique data pointer. The internal organisation of data space is similar to data blocks in Oracle or in any other relational database.

This approach has several advantages:

1. usage of fixed-length cell simplifies cell allocation and automatic storage reclamation
2. it is possible to store many different object models

The second advantage is a more important one. It allows to store different data (class-based objects, prototype-based objects, XML data and even relational data) together in the same database. Thus CellStore can act as a pure object database or as an XML database. Note that data are stored in their native form, mapping of data to cell and data space is less or more straightforward. This is why we call CellStore a universal database.

Mapping objects into cell and data space In this section, mapping of objects will be described. Consider class-based object model and eight-field cells.

Each object occupies at least one cell, called the *head cell* and zero or more cells called *tail cells*.

The head cell contains cell header, which contains cell type (non-indexable class-based instance for example), other information like the number of cells occupied by this instance, gc support information, tail-cell flag and more.

Second field of the head cell contains pointer to ACL set, pointer to another object (in fact, pointer to another object's head cell), which contains all information needed for access control to this object (because we are designing multi-user database).

Third field of the head cell contains pointer to object's class, which is also an object represented by head cell.

Other fields contain pointers to ordinary instance variables. If all instance variables cannot be stored in a single cell, the last field contains pointer to the next (possibly tail) cell. Another possibility is to use something like indirect pointers as used in inode-based file systems.

Data of indexed classes (arrays, byte arrays) are stored in the data space.

An example of objects structure and its mapping to cell and data space is on figure 1 and figure 2. Note, that integers are stored as immediate values [1].

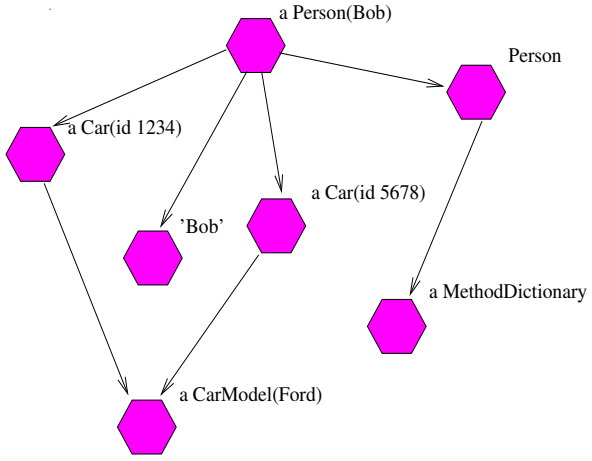

Fig. 1. Example of object structure

Mapping XML data into cell and data space Another example of data that can be stored in CellStore is XML data. Although XML data can be stored into CellStore as normal objects (DOM nodes) as shown above, we are using more efficient, XML specific mapping.

There are 9 types of cells:

- character data cell
- attribute cell
- element cell
- document cell
- document type cell
- processing instruction cell
- comment cell
- xml resource cell
- collection cell

The last two cell types represent XML:DB objects as described in [2]. Each cell has a pointer to its parent cell, first child cell and sibling cell. Meanings of the last four fields depend of the type of the cell (see table 1).

Children of any cell are linked through the sibling pointer and parent holds pointer to the first child.

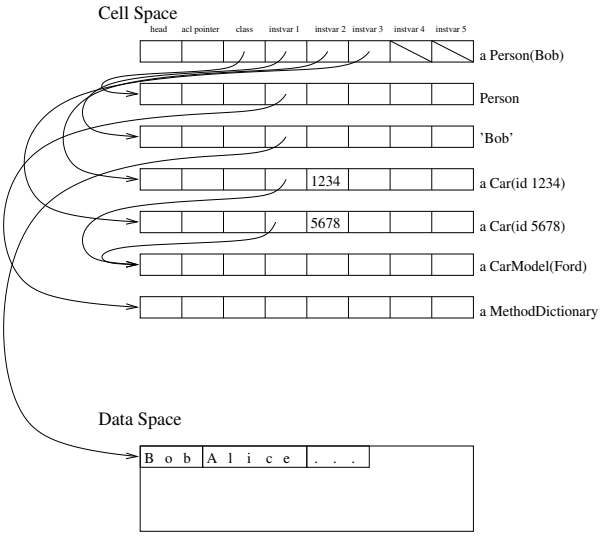

Fig. 2. Example mapping objects into the cell and data space

3.2 The CellStore's virtual machine

Classic virtual machine consists of an object memory and an interpreter. Object memory is responsible for managing objects in memory, for efficient storage reclamation and the interpreter defines all the execution semantics. We think that it's possible to implement virtual machine on the top of CellStore storage, so one can think about CellStore as one large multi-user virtual machine with persistent, transaction-capable object memory.

The idea is to move as much functionality as possible to CellStore's virtual machine. This includes indexing algorithms, garbage collector, jitter etc. The CellStore should provide only basic object memory management and common,

Table 1. Meanings of fields in XML cells

	Field			
Cell type	5	6	7	8
character data	data 1	(data 2)	(data 3)	(data 3)
attribute	local name	namespace qualifier	namespace uri	value
element	local name	namespace qualifier	namespace uri	first attribute
document	document type	encoding	unused	unused
document type	public id	system id	unused	unused
processing instruction	target	data	unused	unused
comment	data 1	(data 2)	(data 3)	(data 3)
xml resource	resource name	unused	unused	unused
collection	name	unused	unused	first resource

flexible object model. Everything else could be implemented on the top of CellStore.

This allows user (programmer) to experiment with different algorithms, jitters, garbage collectors and, as long as the interpreter itself will be implemented on the top of CellStore, with different programming languages and code semantics.

The CellStore's virtual machine should provide only the following:

- object memory management supporting only common object model as described in section 3.1
- dumb, built-in interpreter which is capable if interpreting simple, limited language (bytecode) – we called it the *bootstrap interpreter*
- capability of trap out unknown language (bytecode) and let user-level interpreter to evaluate them.
- basic support for installing native (jitted) code into VM's native code cache

There is no need for speed of any interpreter as long as the interpreter will be able to interpret jitter (implemented in any language). The jitter can translate itself into the native code to make itself fast and then translate the rest.

3.3 Architecture of CellStore database

The high level architecture of CellStore is shown on figure 3.

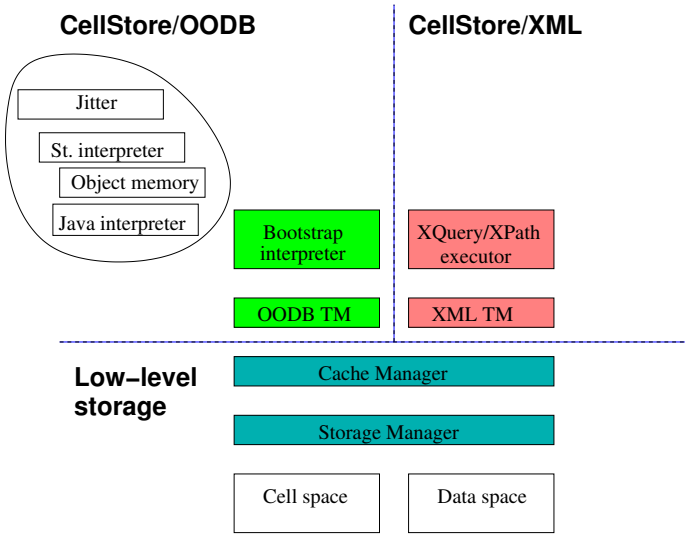

Fig. 3. High level architecture of CellStore

From the VM's side of view, OODB transaction manager plays the role of object memory, so it should provide interface similar to Smalltalk-80's object

memory [1]. In addition, it must provide an interface for transaction managing (start, commit, abort) and an interface for garbage collector.

3.4 Status of the CellStore project

The Cellstore project is developed at Department of Computer Science, FEE CTU Prague by Michal Valenta, Jan Vrany, Pavel Strnad, Karel Prihoda and Jan Zak.

Whole the project is developed in Smalltalk/X – a free smalltalk implementation. Smalltalk/X has been chosen because of its pure object orientation, source code availability, outstanding development tools and because of its extreme agility. To achieve practical performance, system can be translated to C [4].

In these days, only the lowest level storage manager is implemented. It can manage cell and data spaces. First experiments show that the storage is able to store whole INEX database [10] (about 500MB of XML documents) using mapping described in section 3.1 without significant performance lost, that means that the document reconstruction time of single, randomly chosen document n was almost independent on database size.

First versions of cache and XML transaction managers are implemented and tested but they are not integrated to the rest of the system, yet.

4 Conclusion and future work

This paper presented the vision of a pure object database built on the top of CellStore storage model. In CellStore virtual machine, as much components as possible is lifted up to "user-space", making experiments with different languages, semantics, jitter, garbage collectors and other algorithms and techniques very easy.

To make such system working, several things has to be developed:

- OODB transaction manager and its interface to bootstrap interpreter.
- tiny bootstrap interpreter
- experimental, naive one-to-one non optimising jitter
- other language interpreter

Once things mentioned above will be implemented and tested, we will have a working database system, which can be used as test bed for many different algorithms. Such system will make development of new approaches and algorithms very easy.

References

1. A. Goldberg, D. Robson. *Smalltalk-80: The Language and its Implementation*, Addison-Wesley, 1983.

2. XML:DB initiative. *XML:DB Working Draft*, http://xmldb-org.sourceforge.net/xapi/xapi-draft.html
3. Camp Smalltalk. *VM Issues*, http://wiki.cs.uiuc.edu/CampSmalltalk/VM+Issues
4. D. Ingalls, T. Kaehler, J. Maloney, S. Wallace, A. Kay. *Back to the Future. The Story of Squeak, A Practical Smalltalk Written in Itself* http://users.ipa.net/~dwighth/squeak/oopsla_squeak.html
5. Camp Smalltalk. *GLORP: Generic Lightweight Object-Relational Persistence*, http://glorp.org/
6. *Relational Persistence for Java and .NET*, http://www.hibernate.org/
7. *OmniBase*, http://www.gorisek.com
8. *DB4Objects*, http://db4objects.org
9. *GemStone/S*, http://www.gemstone.com
10. *INEX: Initiative for the Evaluation of XML Retrieval*, http://inex.is.informatik.uni-duisburg.de/2006/

Conceptual Modeling for XML: A Survey*

Martin Nečaský

Charles University, Faculty of Mathematics and Physics,
Malostranske nam. 25, 118 00 Praha 1, Czech Republic
martin.necasky@mff.cuni.cz

Abstract. Recently XML is the standard format used for the exchange of data between information systems and is also frequently applied as a logical database model. If we use XML as a logical database model we need a conceptual model for the description of its semantics. However, XML as a logical database model has some special characteristics which makes existing conceptual models as E-R or UML unsuitable. In this paper, the current approaches to the conceptual modeling of XML data are described in an uniform style. A list of requirements for XML conceptual models is presented and described approaches are compared on the base of the requirements.

Keywords: conceptual modeling, XML, XML Schema

1 Introduction

Today XML is used for the exchange of data between information systems and it is frequently used as a logical database model for storing data into databases. If we use XML as a logical database model we need a conceptual model for modeling XML data. There is the Entity-Relationship (E-R) [19] model for the conceptual modeling of relational data. However, XML as a logical database model has some special differences which makes the E-R model unsuitable for the conceptual modeling of XML data. The main differences are the following:

- hierarchical structure
- irregular structure
- ordering on siblings
- mixed content

These features cannot be properly modeled in the E-R model. There are some approaches, for example Extended E-R [1], EReX [11], EER [12], XER [17], ERX [16], and C-XML [5], trying to extend the E-R model to be suitable for the conceptual modeling of XML data. It is possible to extend the E-R model to model ordering, mixed content and irregular structure of XML data. However, there is a problem with the modeling of a hierarchical structure of XML data.

* This paper was supported by the National programme of research (Information society project 1ET100300419)

Suppose an E-R diagram with a relationship type *Enroll* between two entity types *Student* and *Course* representing courses enrolled by students. Each student may enroll zero or more courses and each course may be enrolled by zero or more students. The diagram is shown in Figure 1(a).

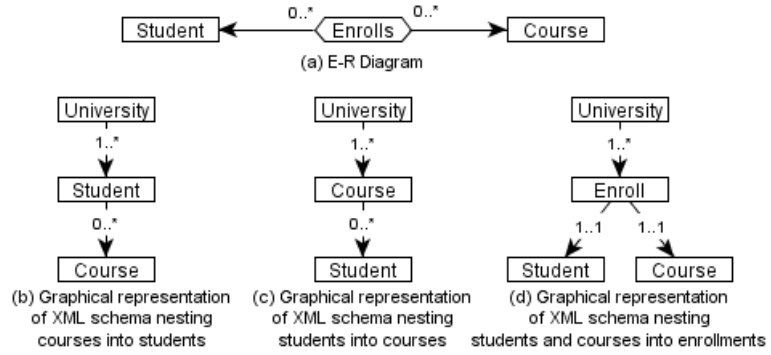

Fig. 1. Representation of E-R relationship type in a hierarchical structure

Figures 1(b), (c), and (d) show possible representations of the relationship type in a hierarchical structure. Oriented arrows denote a nesting. There is not the best nesting of the concepts. The nesting of courses into students illustrated by Figure 1(b) is suitable when we need to see students and the courses they enrolled. The nesting of students into courses illustrated by Figure 1(c) is suitable when we need to see courses and the students enrolled in them.

The previous example shows another difference between the conceptual level of XML and the E-R model. This difference is not in the structure but it is in the usage of XML. It is shown that there may be many ways of how to use entity types connected together by a relationship type. If we represent data in the form of XML, each of these ways may require another hierarchical ordering of the entities. However, this feature cannot be effectively modeled by the E-R model.

Another possibility of how to model XML data is to start from a hierarchical structure. This approach may be called the hierarchical approach. There are conceptual models based on the hierarchical approach, for example X-Entity [10], ORA-SS [4], and Semantic Networks for XML [6]. The base of a schema in the hierarchical approach is a tree, whose nodes are entity types and edges are relationship types between entity types. Figures 1(b), (c), and (d) show examples of a basic hierarchical schemata.

The hierarchical approach is able to solve the mentioned problem with different views of the same data. For each of the views there is a separate tree. However, a problem with the modeling of attributes of relationship types or with the modeling of n-ary relationship types, effectivelly solved in the E-R model,

arises. Another problem arises when deciding which of hierarchical organizations of the same data is the best to select as the basic organization used for the data storage.

The goal of this paper is to describe the existing conceptual models for XML based on the E-R model and on the hierarchical approach. There are approaches based on the UML (Unified Modeling Language) [15] and ORM (Object Role Modeling) [8] models, too. However, we do not describe them in this paper. We propose a list of requirements for conceptual models for XML and compare the described models against the requirements. The main contributions of this paper are the unified descriptions of the conceptual models and the comparison of the models against the list of requirements. This paper is an abbreviated version of the full paper [14] where all the conceptual models mentioned in this paper are compared in detail and described in an unified formalism.

Section 2 introduces the list of requirements for conceptual models for XML. Section 3 describes representatives of the conceptual models for XML based on the well-known E-R model. Section 4 describes representatives of the hierarchical conceptual models for XML. Section 5 compares the described conceptual models against the requirements introduced in Section 2.

2 Requirements for Conceptual Models for XML

Requirements for conceptual models for XML are summarized in this section. There are two groups of the requirements described. The first group consists of *general requirements* covering general goals of the XML conceptual modeling. The second group consists of *modeling constructs requirements* covering requirements on what kinds of modeling constructs should XML conceptual models support.

2.1 General Requirements

Independence on XML schema languages The conceptual model should be independent on a certain XML schema language (XML Schema [7], DTD, ...). The constraints given by a certain XML schema language should not be propagated to the conceptual level.

Formal foundations The modeling constructs of the conceptual model should be described formally, which allows to compare the model with other conceptual models or to describe the operations on the model structures and modeled data (for example, data transformation between two conceptual schemata or their integration).

Graphical notation A user-friendly graphical notation for the formal modeling constructs should be offered by the conceptual model.

Logical level mapping There should be algorithms for mapping of the conceptual modeling constructs to the XML logical level. The logical schema should implement as many integrity constraints arised from the conceptual schema as possible. It may require the usage of more than one XML schema

language for the logical level description (XML Schema and Schematron [9], for example). The hierarchical structure of the XML data should be utilized as much as possible on the logical level.

Different structures on the logical level The XML logical level is hierarchical. However, there are different users with different requirements accessing the modeled data on the logical level. Hence, there can be different hierarchical views of the same data. Each of the views suits to different requirements. It should be possible to model the different hierarchical views on the conceptual level and translate them to the corresponding views on the logical level. Moreover, there should be algorithms allowing automatic translation of data from one logical view to another logical view (using XSLT [2], for example).

Semantic web mapping With the increasing usage of the semantic web technologies the problem of publishing data in the form of RDF [13] triples described by RDF Schema [13] or OWL [18] arises. One possible solution is to have the data internally represented in the form of XML and translate them to the RDF triples represented in the form of RDF/XML [13] utilizing XSLT. The conceptual model for XML should consider this problem. It would be useful to have algorithms for the translation from the conceptual level to the semantic web level where the structures from the conceptual level are described using OWL. It would allow companies to publish their internally represented data on the semantic web and, backwards, to obtain data from the semantic web and integrate them to the internal representation automatically.

2.2 Modeling Constructs Requirements

Hierarchical structure Although it can be useful to keep a document designer out of the hierarchical structure of XML data on the conceptual level, the conceptual model should offer modeling constructs for modeling nesting explicitly. For example, aggregation relationship types can be used. However, non-hierarchical relationship types (for example, association relationship types or references) should be offered too. The conceptual model should introduce contructs for modeling a recursive structure.

Cardinality for all participants The hierarchical structure of XML data restricts the specification of cardinality constraints only to the nested participants of the relationship type. However, it should be possible to specify cardinality constraints for the all participants on the conceptual level.

***N*-ary relationship types** For the same reason, the modeling of n-ary relationship types and their translation to the XML logical level is problematic. However, it should be possible to model n-ary relationship types on the conceptual level.

Attributes of relationship types For the same reason again, the modeling of attributes of relationship types is problematic. Nor the nesting nor the concept of referential integrity on the XML logical level do not allow to

directly express attributes of relationship types. However, the conceptual model should allow to model attributes of relationship types.

Ordering XML is ordered and this property should be propagated to the conceptual level. It should be possible to express the ordering on values of attributes, the ordering on concepts connected with another concept (for example, a book has a title page first, followed by an abstract, chapters, appendixes and a bibliography in this order) and the ordering on a participant of a relationship type (for example, the list of authors of a book or the list of chapters of a book are ordered).

Irregular and heterogeneous structure XML data may have irregular and heterogeneous structure. The conceptual model should introduce constructs for modeling such a structure. For example, variant-valued constructors for constructing attributes or disjunctive relationship types should be introduced.

Document-centric data The difference between the conceptual models for XML and the other conceptual models is that the conceptual models for XML must allow to model document-centric data. It means that not only the real-world objects with attributes and relationships but also the certain parts of documents are modeled on the conceptual level. Hence, there should be corresponding modeling constructs offered by the conceptual model. It means to allow attributes and relationships of a given concept to be mixed with a text when represented in a document content. However, the mixed content should not be restricted as it is restricted by XML Schema. Some form of generalized mixed content should be introduced allowing to specify where the text values may appear exactly (as it is possible in Relax NG [3] schemata, for example).

Reuse of content The reuse of content should be supported by the conceptual model. For example, the concept inheritance (modeled by IS-A relationship types in E-R, for example) supports the reuse of content. However, the conceptual model may be inspired in the XML Schema language and may support named types and named groups of concepts on the conceptual level.

Integration of conceptual schemata XML data are often used for the data integration. However, it can not be done effectivelly and automatically without the support on the conceptual level. A conceptual model for XML should offer modeling constructs to support an integration of schemata on the conceptual level and it should allow to merge different conceptual schemata to an overall conceptual schema. Further, it would be useful to generate XSLT transformation scripts to translate data corresponding to one conceptual schema to data corresponding to another conceptual schema.

3 E-R Based Conceptual Models for XML

In this section, we describe two representatives of the conceptual models for XML based on the E-R model. The first representative is the Extended E-R model [1] and the second representative is the EReX model [11].

3.1 Extended E-R Model (by Antonio Badia)

Extended E-R model proposed by Badia in [1] is a minimalistic extension to the E-R model. The extension is based on the idea of integration of structured and semistructured data where an overall conceptual schema is needed. Moreover, the author proposes algorithms for the translation of E-R schemata to relational schemata and to DTD schemata. Further, he studies the utilization of combination of relational schemata and DTD schemata for a data representation. The author proposes the following DTD based extensions to the E-R model.

- Each attribute is marked as *optional* or *required*. If an entity type has an optional attribute its entities may or may not have a value of the attribute. If an entity type has a required attribute its entities must have a value of the attribute.
- A choice between two or more attributes called *choice attribute* can be modeled. A choice attribute can be *inclusive* or *exclusive*. If an entity type has an inclusive choice attribute its entities may have values of one or more of the attributes in the choice. If an entity type has an exclusive choice attribute its entities may have only a value of one of the attributes in the choice.

In a graphical representation, an optional attribute is connected to the corresponding entity type by a solid line with two dashes crossing it. A choice of attributes is expressed by marking the choice with an upward triangle, with the choices in the opposed side of the triangle.

Figure 2 displays the entity type *Student* having the optional attribute *phone* and the exclusive choice attribute involving the attributes *hosteladdr* and *homeaddr*, i.e. each student has a hostel address or a home address but not both.

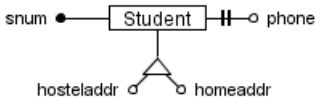

Fig. 2. Extended E-R Diagram

3.2 EReX

EReX is an extension to the E-R model proposed by Mani in [11]. The author introduces the following extensions to the E-R model:

- *Categorization* of entity types can be modeled using *category relationship types*. Category relationship types are a special kind of binary relationship types similar to IS-A relationship types from the well-known E-R model. A

category relationship type is displayed by an arrow with the label CAT going from its category entity type to its categorizied entity type. The difference between the IS-A relationship types and the category relationship types is that a categorized entity type may have an empty key (i.e. an entity type with an empty key must be categorized). Moreover the integrity constraints called *coverage constraints* can be specified on categorizied entity types.

- *Total* and *exclusive coverage constraints* can be specified for categories and for roles of entity types in relationship types. A total coverage constraint specifies that the union of sets of instances of all included categories or roles must be the same as a set of instances of the categorizied entity type or the entity type with the included roles. An exclusive coverage constraint specifies the disjunction between the sets of instances of the included categories or roles. We do not formally define the coverage constraints here. We show them only in a form of the examples illustrated in Figure 3.
- *Order constraints* can be specified for participants of a relationship type. An ordering on a participant E of a relationship type R is displayed by a thick solid line between R and E. If an ordering on E in R is specified, then for a given entity e of the entity type E the set of relationships of the relationship type R with e as a participant is ordered.

The extending modeling constructs of the EReX model are demonstrated by the schema in Figure 3. It displays the categorized entity type $Person$ and its categories $Student$ and $Professor$. The key of $Person$ is empty. Further, there are the entity types $Book$ and $Paper$ connected with $Professor$ by the relationship types $AuthorOfB$ and $AuthorOfP$, respectively. Attributes of $Book$ and $Paper$ are not displayed. There is an ordering specified on the entity types $Book$ and $Paper$ in the relationship types $AuthorOfB$ and $AuthorOfP$, respectively. It means, that the authors of a given paper or a given book are ordered.

The total coverage constraint $Student + Professor = Person$ specifies, that each person is a student or a professor and there are no other persons. The exclusive coverage constraint $Student|Professor$ specifies that students and professors are disjoint. The total coverage constraint $AuthorOfB.pbook + AuthorOfP.ppaper = Professor$ specifies that each professor is an author of some paper or book.

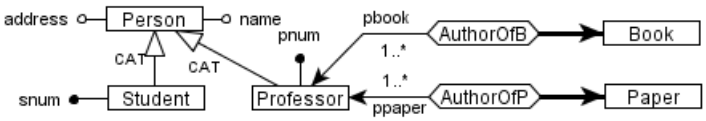

Fig. 3. EReX Diagram

4 Hierarchical Conceptual Models for XML

The extensions of the E-R model allow to model conceptual schemata with a graph structure. However, XML schema languages allow to express relationship types only by nesting and references. It is possible to express all the relationship types from an E-R schema by references, but it leads to flat schemata and the advantages of the hierarchical structure of XML are not utilized. On the other hand, if the hierarchical structure is used to express relationship types in a conceptual schema the problem with the decision about what to nest arises. Another problem is how to represent n-ary relationship types and attributes of relationship types.

In this section we describe a basic hierarchical conceptual model for XML first. In the next subsections, we describe two representatives of the conceptual models for XML based on the hierarchical approach. The first representative is the ORA-SS model [4] and the second representative is the Semantic Networks for XML model [6].

4.1 Basic hierarchical conceptual model for XML

The basic hierarchical conceptual model for XML can be easily defined as a restriction of the E-R model where only the binary relationship types with cardinality types $(1,1):1$ or $(1,1):N$ and without attributes are allowed. Each relationship type is oriented from the entity type with the arbitrary cardinality called the *parent participant* to the entity type with the cardinality $(1,1)$ in the relationship type called the *child participant*. We say that the child participant is *nested* in the parent participant. This kind of relationship types may be called *nesting binary relationship types*. When modeling XML data, the nesting binary relationship types are represented by a nesting of elements on the XML logical level. They express a hierarchical structure on the XML logical level explicitly on the conceptual level. However, the semantics of nesting relationship types do not have to be only "part-of". It may be a general association too.

Such restrictions are too strong and do not allow to model conceptual schemata with richer semantics. Nor n-ary relationship types, nor attributes of relationship types can be modeled. Moreover, lots of redundancies may appear in schemata. There are some approaches extending this basic hierarchical model described in the following subsections.

4.2 ORA-SS

ORA-SS is a rich hierarchical conceptual model for XML proposed by Dobbie et al. in [4]. ORA-SS has three basic modeling constructs: object types, relationship types and attributes. The object type construct is similar to the entity type from the E-R model. Relationship types between object types represent hierarchical relationships. Non-hierarchical relationships can be modeled by the references. The authors introduce the concept of n-ary hierarchical relationship

types and attributes of hierarchial relationship types. Moreover, the authors offer the following extending features:

- *Cardinality constraints* for the both participants of hierarchical relationship types.
- An *ordering* on different concepts. The first type is an ordering on values of a multivalued attribute of an object type. The second type is an ordering between the attributes of an object type and nesting relationship types going from the object type. The third type is an ordering on a relationship type going from an object type. It allows to specify ordering between the objects nested by the ordered relationship type in the parent object.
- A *disjunction* between two or more attributes or nesting relationship types. It allows to model irregular structure.

Figure 4 displays an ORA-SS schema representing professors as employees of departments and professors as members of projects. Each professor is employed by exactly one department and each department employes one or more professors. Each professor is a member of zero or more projects and each project has one or more members. There is a ternary relationship type $AuthorOf$ between the object types $Project$, $Professor$ and $Paper$. It represents papers written by a professor participating in a project. Each professor is an author of zero or more papers in a project and each paper has one or more authors. For each project there is a list of member professors and for each member professor there is a list of papers he wrote during his work in the project. Moreover, there is the attribute *pages* of the relationship type $AuthorOf$. For a professor being an author of a paper in a project the value of *pages* is the number of pages the professor wrote in the paper. However, the attribute *pages* cannot be directly assigned to the relationship type $AuthorOf$. It must be assigned to the nested object type $Paper$. For each $Professor$ instance nested in a $Project$ instance there must be a $Professor$ instance nested in a $Department$ instance containing the *name* and *address* values of the professor. This is modeled by the reference between the object types.

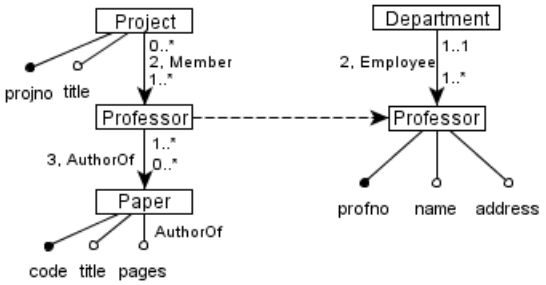

Fig. 4. ORA-SS Diagram

4.3 Semantic Networks for XML

The semantic network model for XML was introduced by Feng et al. in [6]. The model is a little extension to the basic hierarchical conceptual model described in Section 4.1. Schemata in the semantic network model for XML are called semantic networks. Nodes in semantic networks are used for modeling objects from the real world and their attributes, and edges are used for modeling relationships between the objects.

Only binary hierarchical relationship types without attributes can be modeled in the semantic network model. Moreover the parent participant cardinality constraint of a hierarchical relationship type must be equal to $(1, 1)$. Beside the hierarchical relationship types it is possible to use non-hierarchical relationship types for modeling associations.

Different constraints can be specified in the semantic network schema for XML. Constraints can be specified over a node, over an edge and over a set of edges. These constraints are a uniqueness, order, disjunction, etc.

Figure 5 displays a semantic network schema. There are departments represented by the node *Department* and professors in the departments represented by the node $Professor$. The content of the node $Professor$ is ordered. For each professor there are the papers he wrote represented by the node *Paper*. Each paper may be composed of chapters or sections, but not both (the exclusive constraint). The courses offered by a department are represented by the node $Course$. $Professor$ is associated with $Course$. It represents the relationships between a professor and the courses he teaches.

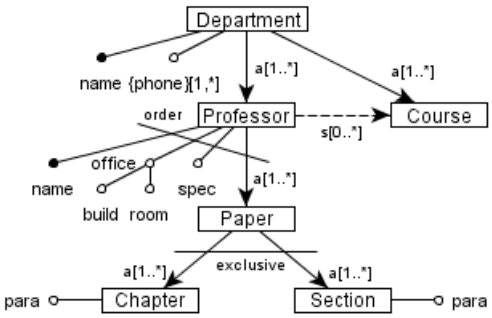

Fig. 5. Semantic Network Diagram

5 Comparison of Described Conceptual Models

In this section, we compare the conceptual models mentioned in this paper. The comparison is made against the general requirements and the modeling constructs requirements introduced at the beginning of the paper.

There are two comparative tables. Table 1 compares the models against the general requirements and Table 2 compares the models against the modeling constructs requirements. The well-known E-R model and the basic hierarchical model are compared too.

We are not able to decide, which of the previous two approaches (E-R extensions, hierarchical modeling) is better for the conceptual modeling of XML data. Conceptual models based on the E-R model allow user to create a schema with no metadata redundancy, but there is the problem with the modeling of the specific XML features. Hierarchical conceptual models solve the problem with a hierarchical structure of XML, but there arises problems such as data and metadata redundancy, modeling of attributes of relationship types, and modeling of n-ary relationship types.

There are requirements that are not met by the described models. The modeling of document centric data and the reuse of content is problematic. The important requirement on the integration of conceptual schemata is solved only by the ORA-SS model. None of the models solves the problem of the integration with the semantic web technologies.

Table 1. Comparison Against the General Requirements

ER	ER-B	EReX	EER	XER	ERX	C-XML	Hier	X-Entity	ORA-SS	Sem.net.
· independence on XML schema languages										
√	–	√	√	–	√	√	√	√	√	√
· formal foundations										
√	√	√	–	–	√	–	√	√	√	√
· graphical notation										
√	√	√	√	√	√	√	√	√	√	√
· logical level mapping										
– relational model										
√	√	–	–	–	–	–	–	–	–	–
– tree grammar based XML schema languages										
–	√	√	√	√ 1	–	√	√	√	√	√
– pattern based XML schema languages										
–	–	–	–	–	–	–	–	–	–	–
– utilization of hierarchical structure of XML										
–	–	√	√	√	–	√	√	√	√	√
· different structures on the logical level										
– conceptual hierarchical views										
–	–	–	–	–	–	–	–	–	√	–
– translation between hierarchical views										
–	–	–	–	–	–	–	–	–	√	–
· semantic web mapping										
–	–	–	–	–	–	–	–	–	–	–
1 formal description is missing										

Table 2. Comparison Against the Modeling Constructs Requirements

ER	ER-B	EReX	EER	XER	ERX	C-XML	Hier	X-Entity	ORA-SS	Sem.net.
· hierarchical relationship types										
–	–	–	√	–	√	–	√	√	√	√
– $M:N$ cardinality – N-ary – attributes										
–	–	–	–	–	–	–	–	–	√	–
· non-hierarchical relationship types										
√	√	√	√	√	√	√	–	–	√	√
– $M:N$ cardinality										
√	√	√	√	–	√	√	–	–	– 5	–
– N-ary – attributes										
√ 1	√	√	√	–	–	√	–	–	– 5	–
· ordering										
– on the values of an attribute										
√ 1	–	–	–	–	–	–	√	–	√	√
– on the content of a concept										
–	–	–	–	√ 2	–	√	–	–	√	√
– on the participant of a relationship type										
–	–	√	√	–	√ 3	–	–	–	√	–
· irregular and heterogeneous structure										
– variant-valued attribute constructor										
√ 1	√	–	–	–	–	–	√	–	√	√
– disjunctive constraints on relationship types										
–	√ 4	√ 4	–	–	√	–	–	√	√	√
· document-centric data										
– basic mixed content										
–	–	–	–	√	–	–	–	–	–	–
– generalized mixed content										
–	–	–	–	–	–	–	–	–	–	–
· reuse of content										
– IS-A or the category concept										
√	√	√	–	√	√	√	√	–	√	√
– named types and groups of concepts										
–	–	–	–	–	–	–	–	–	–	–
· integration of conceptual schemata										
– modeling constructs										
–	–	–	–	–	√	–	–	–	√	–
– algorithms for merging schemata										
–	–	–	–	–	–	–	–	–	√	–
– algorithms for the data translation between schemata										
–	–	–	–	–	–	–	–	–	√	–
1 with the complex attributes extension										
2 unordered content is restricted by the restrictions of xsd:all										
3 only by ordered attributes, native XML ordering is not utilized										
4 using the category concept										

6 Conclusions

In this paper, we describe a state of the art of the conceptual modeling for XML. There has been several papers proposing new conceptual models for XML. We selected some representants of the models and describe them in this paper. We propose a list of requirements conceptual models for XML should satisfy and compare the mentioned conceptual models against the requirements.

The comparison of the models shows that there is a poor support for some specific XML features as ordering or mixed content as described by the modeling constructs requirements proposed in Section 2. Moreover, the models poorly concentrates on the usage of conceptual schemata for the data integration and the integration with the semantic web technologies as described by the general requirements proposed in Section 2.

For these reasons, there is an open space for a research in the area of the conceptual modeling for XML. Not only new modeling constructs should be proposed to support specific XML features. The utilization of the conceptual models for the data integration between different data sources including semantic web resources should be explored too.

References

1. A. Badia. Conceptual Modeling for Semistructured Data. In *Proceedings of the 3rd International Conference on Web Information Systems Engineering Workshops (WISE 2002 Workshops), p. 170-177.* Singapore, December 2002.
2. J. Clark. *XSL Transformations (XSLT) Version 1.0.* World Wide Web Consortium, Recommendation REC-xslt-19991116. November 1999.
3. J. Clark, M. Makoto. *RELAX NG Specification.* Oasis. December 2001.
4. G. Dobbie, W. Xiaoying, T.W. Ling, M.L. Lee. ORA-SS: An Object-Relationship-Attribute Model for Semi-Structured Data. *Technical Report, Department of Computer Science, National University of Singapore.* December 2000
5. D.W. Embley, S.W. Liddle, R. Al-Kamha. Enterprise Modeling with Conceptual XML. In *Proceedings of the 23rd International Conference on Conceptual Modeling (ER 2004), p. 150-165.* Shanghai, China, November 2004.
6. L. Feng, E. Chang, T. Dillon. A Semantic Network-Based Design Methodology for XML Documents. *ACM Transactions on Information Systems, Volume 20, Number 4, p. 390-421.* October 2002.
7. D. C. Fallside, P. Walmsley. *XML Schema Part 0: Primer Second Edition.* World Wide Web Consortium, Recommendation REC-xmlschema-0-20041028. October 2004.
8. T. Halpin. *Information Modeling and Relational Databases From Conceptual Analysis to Logical Design.* Morgan Kaufmann Publishers, 2001. ISBN: 1-55860-672-6
9. International Organization for Standardization. *Information Technology Document Schema Definition Languages (DSDL) Part 3: Rule-based Validation Schematron.* ISO/IEC 19757-3, February 2005.
10. B.R. Loscio, A.C. Salgado, L.R. Galvao. Conceptual Modeling of XML Schemas. In *Proceedings of the Fifth ACM CIKM International Workshop on Web Information and Data Management (WIDM 2003), p. 102-105.* New Orleans, Louisiana, USA, November 2003.

11. M. Mani. EReX: A Conceptual Model for XML. In *Proceedings of the Second International XML Database Symposium (XSym 2004), p. 128-142*. Toronto, Canada, August 2004.
12. M. Mani, D. Lee, R.R.Muntz. Semantic Data Modeling Using XML Schemas. In *Proceedings of the 20th International Conference on Conceptual Modeling (ER 2001), p. 149-163*. Yokohama, Japan, November 2001.
13. F. Manola, E. Miller. *RDF Primer*. World Wide Web Consortium, Recommendation REC-rdf-primer-20040210. February 2004.
14. M. Necasky: Conceptual Modeling for XML: A Survey. Technical Report No. 2006-3, Dep. of Software Engineering, Faculty of Mathematics and Physics, Charles University, Prague, 2006, 54 p.
15. Object Management Group. *UML 2.0 Superstructure Specification*. October 2004.
16. G. Psaila. ERX: A Conceptual Model for XML Documents. In *Proceedings of the 2000 ACM Symposium on Applied Computing, p. 898-903*. Como, Italy, March 2000.
17. A. Sengupta, S. Mohan, R. Doshi. XER - Extensible Entity Relationship Modeling. In *Proceedings of the XML 2003 Conference, p. 140-154*. Philadelphia, USA, December 2003.
18. M. K. Smith, Ch. Welty, D. L. McGuinness. *OWL Web Ontology Language Guide*. World Wide Web Consortium, Recommendation REC-owl-guide-20040210. February 2004.
19. B. Thalheim. *Entity-Relationship Modeling: Foundations of Database Technology*. Springer Verlag, 2000, Berlin, Germany. ISBN: 3-540-65470-4

Transforming Data from DataPile Structure into RDF

Jiří Dokulil

Charles University, Faculty of Mathematics and Physics
Malostranské nám. 25, 118 00 Praha 1, Czech Republic
dokulil@gmail.com

Abstract. Huge amount of interesting data has been gathered in the DataPile structure since its creation. This data could be used in the development of RDF databases. When limited to basic information stored in the DataPile the transformation into RDF is straightforward. It still provides millions of RDF triples with complex structure and many irregularities.

1 Introduction

While it is easy to find huge relational or XML data rich in structure there is still not much data available in the RDF format. Such data could be obtained by simple conversion from a relational database but this data would be simple and with a regular structure.

In this paper we propose a transformation of data stored in the DataPile structure [1] into the RDF [2]. We expect to receive huge amount of RDF triples with more interesting structure and much less regular than data from relational databases. This expectation is based on the way the DataPile structured is being used in practice.

The DataPile system was developed to integrate data from a heterogeneous set of databases. Among main design goals were storage of historical versions of data and easy adaptation to global schema changes.

First of all we present the DataPile and RDF models, then describe the transformation of metadata and data and finally give results of an experimental implementation of the transformation.

2 The Data Models

In this section we present the data models that take part in the transformation.

2.1 The DataPile

The terminology used in DataPile systems is different from those used in relational and RDF databases. *Entity* is a rough equivalent of a table scheme. It has a name and consists of *attributes*, which can be compared to column definitions. Each attribute defines an attribute name and data type. The set of allowed data types had to be very

limited because of implementation reasons. The only supported types are string, number, timestamp, BLOB (Binary Large Object) and a typed reference (foreign key). The DataPile system allows definition of multiple entities each having zero or more attributes. One attribute can not be a member of multiple entities. On the other hand, multiple attributes with the same name can exist, as long as they are members of different entities.

Entities and attributes are *metadata*. They define structure of the actual data that can be stored in the system.

The data consist of *attribute values*. An attribute value is one data item together with type information (identifier of an entity attribute), validity period, relevance and source of the value. Attribute values describing one object are grouped together into an *entity instance*. Each entity instance is assigned a unique eighteen-digit number called *entity instance* identifier. Only attributes of one entity can be used as types of attribute values forming one entity instance. This entity is called *a type of* that *entity instance*.

All entity instances of one type can be viewed as a relational table with each row containing one entity instance. Then one attribute value would be one item of this table.

All this information (matadata and data) is stored in a relational database with special schema called the DataPile structure. This structure and a set of applications and tools form the DataPile system.

In order to achieve the goals set for the DataPile the data could not be stored using one table for each entity type. Instead, a special DataPile structure was created. The center of the structure is one table called PILE capable of storing all attributes of all entities along with their history was used. This table is supplemented by other tables that store metadata, e.g. list of entities and their attributes.

One row of the PILE table contains entity instance identifier, attribute identifier, attribute value, validity period, and other information used in the data integration process. The attribute value is stored in more table columns. It requires one column for each data type. This is the reason why only fixed and very limited set of data types was allowed in the DataPile structure.

Let us look at an example. Consider a system for storing basic information about people. Relational schema could look like this:

```
PERSON(id, first_name, last_name, date_of_birth).
```

In a data pile system this schema would require metadata containing one entity called “PERSON” consisting of three attributes (first_name, last_name and date_of_birth). Data type of first_name and last_last name would be a string in both models. On the other hand there is no exact equivalent for “date” data type, which would probably be the type of the date_of_birth column in a relational database. The “timestamp” data type would have to be used.

Table 1. Example data to be stored in the DataPile

id	First_name	last_name	date_of_birth
1	John	Smith	5.8.1962
2	Jane	Doe	23.2.1971

We can now transform relational data from the Table 1 into the DataPile structure. First of all, both records have to be assigned an entity instance identifier. Normally it would have been an eighteen-digit number but for convenience we use 101 and 102 as the identifiers.

Two instances of entity "PERSON" with identifier 101 and 102 have to be created. Then the appropriate attribute values are to be created in the PILE table. PILE table containing these attributes is displayed in the Table 2.

The table also contains an example of storing historical version of data. On 5.7.2005 the name of Jane Doe was changed to Joan Doe.

Table 2. PILE table with example data (simplified, some columns omitted). Ent_id stands for entity instance identifier.

ent_id	attribute	string value	time value	valid from	valid to
101	first_name	John	*null*	28.5.2005 15:31:20	*null*
101	last_name	Smith	*null*	28.5.2005 15:31:20	*null*
101	date_of_birth	*null*	5.8.1962 0:00:00	28.5.2005 15:31:20	*null*
102	first_name	Jane	*null*	27.5.2005 10:12:25	5.7.2005 9:25:05
102	first_name	Joan	*null*	5.7.2005 9:25:05	*null*
102	last_name	Doe	*null*	27.5.2005 10:12:25	*null*
102	date_of_birth	*null*	23.2.1971 0:00:00	27.5.2005 10:12:25	*null*

2.2 The RDF

One of the goals of the RDF is integration of data gathered about resources on the World Wide Web. Such data tend to be rich in structure and often incomplete.

The RDF is used to make statements about resources. A RDF statement is a triple consisting of a subject, a predicate and an object. This states that the subject has a property (predicate) with a certain value (object). The statement is modeled as a graph with one node for the subject, one node for the object and an arc for the predicate, directed from the subject node to the object node.

A typical example looks like this:

```
<http://example.org/book/book1>
<http://purl.org/dc/elements/1.1/>
"John Smith"
```

This states that the book identified by URI <http://example.org/book/book1> was created by John Smith. The book is the subject, "created" is the predicate and John Smith is the object of the triple. In this example we represent John Smith by a literal

"John Smith". A literal is a constant expression that can be typed or untyped (plain). They are used to represent values like numbers and dates by their lexical representation. It is always possible to use URI instead of a literal, e.g. <http://www.example.org/staffid/8574>. Then we could also make statements about John Smith. Literals can only be used as objects while URI can take any place in a triple.

URIs are represented by named nodes in the RDF graph. However we do not always need direct access to every node in the graph. Some nodes are always accessed using arcs from other nodes. These nodes do not need universal identifiers like URIs. They can be created as *blank nodes*. These nodes can be used as subjects and objects. Blank nodes are usually assigned a unique identifier when the graph is serialized to a triples representation. Common way of writing such identifiers is _:*identifier*, e.g. "_:blank123". This identifier represents the same blank node in the whole representation of the graph. Different identifiers represent different blank nodes.

3 The Transformation

The basic idea behind the transformation is that by making a projection of the PILE table on the columns containing entity instance identifier, attribute and attribute value we receive a set of triples representing statements very similar to RDF statements.

3.1 The Entity Instance Identifiers

All entity instances in the DataPile are assigned a unique eighteen digit number called entity instance identifier.

We need a way to create nodes with unique names in the RDF graph that will represent the objects we want to make statements about. The entity instance identifier is ideal for this. It can be used either as a part of an URI represented by the node or as an identifier of a blank node if we choose not to give a name to the node. In this paper we describe the latter approach since we wanted to create data that would help in the development of RDF databases and queries containing or returning blank nodes are an important feature of the database we want to test.

If naming of the nodes is required then the transformation process can easily be modified to create nodes with URIs.

3.2 The Metadata

Processing of the data in the DataPile is controlled by metadata that is stored in relational tables. In order for the transformation to work at least some part of the metadata must be stored in the RDF as well.

The most important piece of metadata to transform is attributes of the entities. They serve as predicates (arcs of the RDF graph). The very basic RDF representation of a single attribute looks like this (TURTLE notation [3]).

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-
ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix mt : <http://example.org/stoh/metadata/> .

mt:person__name rdf:type rdf:Property .
```

This defines http://example.org/stoh/metadata/person__name to be an attribute. The original version in the DataPile was an attribute called "name" belonging to an entity called "person". We can represent this information in the RDF as well.

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-
ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix mt : <http://example.org/stoh/metadata/> .

mt:person rdf:type rdfs:Class .
mt:person__name rdf:type rdf:Property .
mt:person__name rdfs:domain mt:person .
```

Alternatively we could name the attribute only by its name in the DataPile and omit the name of the entity. This would allow us to make queries like "Give me the name of all entity instances that have a name". On the other hand it would complicate type checking of the values. Because of this we chose the more specific names.

With the information about entities we can specify a type (entity) of an entity instance.

```
_:568421369754123695 rdf:type mt:person .
```

The subject of the triple is a blank node with an eighteen digit identifier identical to the entity instance identifier in the DataPile.

3.3 The Data Types

The DataPile uses a limited number of data types for the attributes. They are listed in Table 1 together with their equivalents after the transformation.

Table 3. Data types in the DataPile and after the transformation

string	http://www.w3.org/2001/XMLSchema#string
number	http://www.w3.org/2001/XMLSchema#decimal
timestamp	http://www.w3.org/2001/XMLSchema#dateTime
entity reference	*reference to a blank node*

Using these data types we can extend the transformed metadata representation.

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-
ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
```

```
@prefix mt : <http://example.org/stoh/metadata/> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#>

mt:person rdf:type rdfs:Class .
mt:person__name rdf:type rdf:Property .
mt:person__name rdfs:domain mt:person .
mt:person__name rdfs:range xsd:string .
```

The entity references in the DataPile are typed references. One attribute can only be used to reference one specified entity. This is equivalent to specifying one class as a range of a property.

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-
ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix mt : <http://example.org/stoh/metadata/> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#>

mt:person rdf:type rdfs:Class .
mt:address rdf:type rdfs:Class .
mt:person__address rdf:type rdf:Property .
mt:person__address rdfs:domain mt:person .
mt:person__address rdfs:range mt:address .
```

3.4 Transforming the Data

After storing the necessary metadata in the RDF graph we can start transforming the real data. Since every row of the PILE table contains entity instance identifier, attribute identifier and typed value we can make a simple projection of the PILE table on these columns and create one RDF triple from each row.

An example output could look like this:

```
@prefix xsd : <http://www.w3.org/2001/XMLSchema> .

_:568421369754123695 mt:person__name “John Smith” .
_:568421369754123695 mt:person__date_of_birth
     “1980-08-14T00:00:00”^^xsd:dateTime .
_:568421369754123695 mt:person__height
     “1.82”^^xsd:decimal .
_:568421369754123695 mt:person__father
     _:684258941535789524 .
```

The last triple is a reference to another entity instance (a foreign key).

3.5 Multilingual Attributes

Although the presented general transformation is capable of handling all types of date stored in the DataPile there is one case that could be handled in a better way. Practical application of the DataPile showed that it is sometimes necessary to handle string values that need to be expressed in different languages. For instance name of a department in Czech and English or same word in different cases.

Using the DataPile it was necessary to create two new entities causing this feature to be hard to use. The RDF offers an easier way of achieving the same results. The standard offers a way to specify a language tag for every string literal. The language tags are defined by RFC 3066 [4] which is flexible enough to specify not only language but different cases as well.

```
_:469751359754692454 rdf:type mt:department .
_:469751359754692454 mt:department__name
     “Katedra softwarového inženýrství”@cs .
_:469751359754692454 mt:department__name
     “Department of Software Engineering”@en .

_:954783125769542934 rdf:type mt:place .
_:954783125769542934 mt:place__name
     “Praha”@cs-CZ-singular-nominative .
_:954783125769542934 mt:place__name
     “v Praze”@cs-CZ-singular-locative .
```

3.6 Reification

On of the important features of the RDF is the ability to make statements about statements. This is called reification. It can be used e.g. to specify an author of a statement.

There is no such universal feature in the DataPile. On the other hand the supplementary columns of the PILE table can be viewed as a special case of reification with a fixed set of predicates. The columns contain information about source of the value, its validity period etc.

Unfortunately, expressing reification in RDF is not very compact. It requires using a new blank node and making at least four statements. The identifier of the blank node can be generated from primary key of the PILE table. The primary key contains sequential numeric value.

```
_:568421369754123695 mt:person__name “John Smith” .
_:r65413 rdf:type rdf:Statement .
_:r65413 rdf:subject _:568421369754123695 .
_:r65413 rdf:predicate mt:person__name .
_:r65413 rdf:object “John Smith” .
_:r65413 mt:valid_from “20050703T15:21:49” .
_:r65413 mt:valid_to “20050821T09:35:12” .
```

The example shows a triple stating a name of a person together with triples that give validity period of the statement.

4 The Experimental Implementation

An experimental implementation of the presented transformation has been created and tested on real data.

4.1 Limitations

The implementation does not include direct support for multilingual attributes nor does it support reification.

4.2 The Data

The data for the experiment has been gathered into the DataPile from different information systems at the Charles University in Prague. Variability of these systems provided us with data that have not only complex schema but also greatly vary in their completeness.

Because the implementation does not support reification the data was limited only to records that are considered to be currently valid. Working with historical versions of data requires access to supplementary columns of the PILE table which requires reification. If all of the data was extracted without the supplementary information it would have created multiple attribute values for one attribute of one entity instance without a way to distinguish the valid values from the historical ones.

4.3 The Test Environment

The current implementation of the DataPile uses Oracle Database 10g for storage. The database was running on a dual XEON P4 2.4 GHz with 2GB RAM and SCSI RAID.

The extractor itself was running on a separate machine with four XEON P4 2.5 GHz CPUs with 16GB RAM and SCSI RAID. It accessed the database directly using Oracle Call Interface with thin abstraction layer on top of it.

The performance of the extraction process depends mostly on the performance of the database. Processing of records returned from the database does not require much memory or CPU time.

4.3 The Extraction

The extraction generated a TURTLE file with 26 813 044 RDF triples. We made two runs of the extraction. In the first run the data was sorted by the entity instance

identifier and attribute. The sorting of the data was done by the database system that contains the DataPile. Although it is not required for the transformation to work it can improve performance of further processing of the data and help with debugging.

In the second run the data was not sorted at all.

The sorted version finished in 1738 seconds while the unsorted took 1073 seconds to complete.

5 Conclusion

Even the very basic version of the extraction provided great amount of interesting data. Implementation of a version handling multilingual attributes is planned in the near future.

We plan to use the extracted data in the development of an experimental RDF database that uses a SPARQL language [5]. It will help us test and tune the performance of such database. The data was gathered from systems that are used in practice and so their schema, size and structure represent real requirements of such systems. The test results should tell us how the database would behave when deployed as a basis for large scale information system or a system integrating large heterogeneous data.

References

1. Bednarek D., Obdrzalek D., Yaghob J., Zavoral F.: Data Integration Using DataPile Structure. In proceedings of the 9th East-European Conference on Advances in Database and Information Systems, Tallinn, Estonia, 2005
2. Carroll J. J., Klyne G.: Resource Description Framework (RDF): Concepts and Abstract Syntax, W3C Recommendation, 10 February 2004 http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/
3. Beckett D.: Turtle - Terse RDF Triple Language http://www.dajobe.org/2004/01/turtle/
4. Alvestrand H.: Tags for the Identification of Languages http://www.ietf.org/rfc/rfc3066.txt
5. Prud'hommeaux E., Seaborne A.: SPARQL Query Language for RDF, W3C Working Draft, 23 November 2005 http://www.w3.org/TR/2005/WD-rdf-sparql-query-20051123/

Acknowledgement

This research was supported in part by the National programme of research (Information society project 1ET100300419).

Towards Better Semantics in the Multifeature Querying**

Peter Gurský

Institute of Computer Science, Faculty of Science
P.J.Šafárik University Košice
Jesenná 9, 040 01, Košice, Slovakia
gursky@upjs.sk

Abstract. Nowadays the natural requirement of users is to retrieve the best answers for the criteria they have. To explain, what kind of objects user prefers, we need to know, which values of properties are suitable for the user. We assume that each property is possibly provided by an external source. Current algorithms can effectively solve this requirement, when the sources have the same ordering as the user preferences. Commonly, two users prefer different values of a given property. In this paper we describe how we can consider this feature.

Key words: multifeature querying, top-k objects, aggregation, fuzzy function

1 Introduction

Many times users want to find the best object or top k objects in the possible huge set of objects. The decision which object is better than the other, is based on the properties of the objects. Typical objects are job offer, document, website, picture, book, presentation, conference, hotel, vacation destination etc.

Such objects have typically some properties (attributes). Users can search and decide, which objects are the best using these properties. Such searching is made by a multifeature deciding.

The property of an object is, typically, one of four types. First type of properties is boolean or yes/no property. Examples can be: work at home, if somebody is married, breakfast included, aspect at the sea, Springer proceedings etc. Second type of properties is properties, that are graded in some way to finite number of classes. Typical properties are: number of stars of hotels, quality of an article, level of education etc. Third type is real or integer number, for example: salary, price, number of pages, properties in multimedia databases, date etc. The last type of properties is text. In multifeature querying we can use this kind of attribute to reduce searching space (user could search only IT jobs), especially,

** This work was partially supported by the project 'Štátna úloha výskumu a vývoja "Nástroje pre získavanie, organizovanie a udržovanie znalostí v prostredí heterogénnych informačných zdrojov" prierezového štátneho programu "Budovanie informačnej spoločnosti".'

when such a property is organized in some hierarchical structure e.g. ontology. We can also derive some other properties from a text. These properties should be one of the three previous types. For example from the human first name we can learn a gender, from the name of a town we can find out a distance from a specific location.

Useful condition is to assume that each property is provided by a possibly remote source. For example to find out distances, we can use servers of traffic companies. Information about free places in hotels can be provided by a server, which collects such information. We can say that sources are distributed, also when the information are on the same computer, but stored in different repositories (RDBS, Ontology, files). This condition is suitable also in cases, when we want to combine several search methods and aggregate them to one list of the best objects. In this case the sources are typical input streams. In the rest of the paper we assume that the properties of objects are provided by distributed sources.

The problem is, how to specify, whose objects are the best. First approach to this problem is to use a monotone aggregation function. Ronald Fagin in 1996 introduced "Fagin's algorithm", which solves this problem first time in [6]. Fagin et al. [7] presented "threshold" algorithm that made the search much faster. Güntzer et al. [1] defined "quick-combine" algorithm using first heuristic. Other heuristics was presented by P. Gurský and R. Lencses [10]. M.Vomlelová and P. Vojtáš [9] propose a probabilistical heuristic. All these solutions use two types of accesses - sorted access and random access. The sorted access is a sequential access to a sorted list. Using this type of access the grades of objects are obtained by proceeding through the list sequentially from the top. Random access returns the property value for a specified object. Further papers deal with the situation, when some kind of accesses is slow or impossible. There was defined a "combined algorithm" in [7] that count with the prices of accesses. In the same paper authors propose algorithm NRA (no random access) that does not use the random access. Güntzer et al. [2] present algorithm "Stream-combine" that uses some heuristics. Combination of last two approaches is "3P-NRA" algorithm presented by P. Gurský [13] with a new heuristic holes. All three approaches use only sorted access.

The second way to specify the objects to retrieve is a skyline. In the skyline, there are objects that are pareto optimal. An object is pareto optimal, if it is not dominated by any other object. Object x dominates object y if x has greater or equal score in all properties and is strictly greater in at least one property than y. Skyline was firstly presented by S. Börzsönyi et al. [8]. Authors put a proposed algorithm to the database query processor. First solution in the field of multifeature querying was presented by W. Balke et al.[3]. In [4] authors combine features of skylining and aggregation functions as "multi-objective retrieval". In this approach we can specify several monotone aggregation functions. The final set of objects is a skyline with respect to the values of the aggegation functions. In this case, values of aggegation functions are construed as the properties for a skyline. W. Balke at al. [5] propose an algorithm for the case of weak pareto

optimality. It differs from "normal" pareto optimality by partial ordering on domains of the properties. In this case object x dominates object y, if x is strictly greater in at least one property and there is no property such that y has greater score than x.

P.Gurský and T.Horváth in [12] use induction of generalized annotated programs (IGAP) to learn monotone graded classification described by fuzzy rules. Fuzzy rules play role of the monotone aggregation function. As input for this approach is classification of several objects by a scale from the worst to the best.

All current solutions assume, that the sources send their property information ordered from the best value to the worst value. None of these solutions allow the user to specify, which values of a property are better than the other. In this paper we discuss about the possibility of preference specification and effective retrieval of top k objects.

Imagine that we want to find a hotel in a city and we can decide using the properties "distance from the centre", "price" and "number of stars". The main problem is that we cannot universally set the orderings from the best distance to the worst distance or from the best number of stars to the worst. One user can say that the best is to sleep in the centre in the low quality and cheap hotel, a second one may prefer hotels in the country (far from the centre) and with highest possible number of stars. The other one prefer rather quiet suburb place and accepts (and needs) traveling to the centre, but not very long. So he or she prefers middle values of the distance. It is possible that 90 % of people prefer cheap hotels before the expensive ones, but, for example, if somebody is on the business travel and the accommodation is paid by the company, he or she may prefer luxury expensive hotels.

Such properties with unknown default orderings are quite common, also in the case of other types of objects. To explain the preferences by a user we can use fuzzy functions. The explanation can be done by a different method too. Since the fuzzy function is an explicit assignment, we assume that there is a transformation to fuzzy functions. We will consider four types of fuzzy sets (see Figure 1). On the axis x, there are property values e.g. price or number of stars. On the axis y we have preferences of a user, where 1 means strong preference and 0 means no preference. Imagine that we want to explain our preference to the property "distance from the center". If we prefer hotels in the center, our fuzzy function will look like the fuzzy function A on the figure 1. Fuzzy function B means that we prefer hotels far away from the center, fuzzy function C means our preference for hotels in suburb. Fuzzy function D means that we want to be right in the center or out of the city. Fuzzy function A in the case of the price property means, that we prefer cheap hotels, and in the case of number of stars it means our inclination to low quality hotels (low number of stars).

These 4 types of fuzzy functions were strong enough to use for user specification in all domains we considered. It is quite unusual to say, for example, that we prefer middle values but not exactly in the middle, thus the fuzzy function should have two local maximums and three local minimums. The question, if

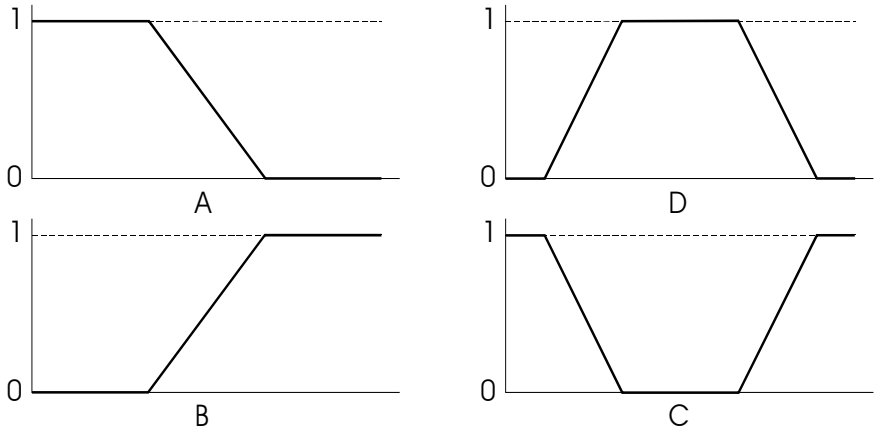

Fig. 1. Fuzzy functions

these 4 types of fuzzy function are enough, is more philosophical. We assume that it is enough.

Interesting way to define user fuzzy functions is to learn them from the evaluation of objects in sample collection. The evaluation can be done in some scale from the best to the worst. It is possible to learn them by the system QUIN (QUalitative INduction). The approach was suggested by Šuc [15].

When the fuzzy functions are defined, the same evaluation of objects can be used by system IGAP to learn monotone graded classification. If we will be able to find top k objects using these fuzzy functions the system presented in [12] should lead to better results. The technique presented in [12] uses only linear regression to learn fuzzy functions so it works good if a user has preferences of properties described by fuzzy functions of type A and B from Figure 1.

In this paper we try to propose a way to extend the functionality of multifeature querying. In the next sections we describe two main approaches to the distributed multifeature querying. Then we propose new extensions of these algorithms.

2 Model

First of all we need to specify basic model and determine useful terms. Assume we have a finite set of objects. Cardinality of this set is n. Every object x has m attributes $x_1, \ldots, x_m$. All objects (or identifiers of objects) are in lists $L_1, \ldots, L_m$, each of length N. Objects in list L_i are ordered descending by values of attribute x_i. As we said, we can define two functions to access objects in lists. Let x be an object. Then $r_i(x) = x_i$ is a grade (or score, rank) of object x in the

list L_i and $s_i(j)$ is an object in the list L_i at the j-th position. Using the function $r_i(x)$ we can realize the *random access* to the lists, i.e. for object x we retrieve the value of i-th property. The second type of access we will use, is the *sorted access* or sequential access to a sorted list. Formally we can say, that sorted access can be described by function $r_i(s_i(j))$. Using this type of access the grades of objects are obtained typically by proceeding through the list sequentially from the top. Let's assume that we have also a monotone aggregation function F, which combines grades of object x from lists $L_1, \ldots, L_m$. We denote the overall value of an object x as $S(x)$ and it is computed as $F(r_1(x), \ldots, r_m(x))$.

Our task is to find top k objects with highest overall grades. We also want to minimize time and space. It means that we want to use as low sorted and random accesses as possible.

We will discuss two main cases. In the first case an algorithm will use both sorted and random accesses, and in the second case we will permit only sorted access. We will show one generalized algorithm for each case. Then we will try to adapt these algorithms to the user preference specifications. Note that all proposed methods can be easily modified for the use of skyline or monotone graded classification.

3 Generalized Threshold algorithm (TA) - uses both random and sorted access

First version of this algorithm was proposed by R. Fagin [7]. Generalized version was published in Gurský et al. [10, 11].

For each list L_i, let $u_i = r_i(s_i(z_i))$ be the value of the attribute of the last object seen under sorted access, where z_i is the position in the list L_i. Define the threshold value τ to be $F(u_1, \ldots, u_m)$. We assume that we have a monotone aggregation function F and the lists are sorted descend by their values from the best to the worst. During the execution of an algorithm we retrieve values from the lists form the greatest to the smallest, thus the threshold τ is the value, which none of still unseen objects can reach [7]. Hence when all objects in the top k list have their property values greater or equal than the threshold, then this top k list is the final and there is none unseen object with greater value. This property is very important to have the algorithm correct.

Let $z = (z_1, \ldots, z_m)$ be a vector, which assigns for each $i = 1, \ldots,$ m the position in list L_i last seen under sorted access. Let H be a heuristic that decides which list (or lists) should be accessed next under sorted access. Moreover, assume that H is such, that for all $j \leq m$ we have $H(z)_j = z_j$ or $H(z)_j = z_j+1$ and there is at least one $i \leq m$ such that $H(z)_i = z_i+1$. The set $\{i \leq m : H(z)_i = z_i + 1\}$ we call the set of candidate lists (or simply candidates) for the next sorted access.

The generalized Threshold algorithm is as follows:

0. Set z:=(0,. . . ,0)
1. Set the heuristic H and do the sorted access in parallel to each of the sorted lists to all positions where $H(z)_i = z_i+1$. Put $z_i = H(z)_i$.

2. First check: Compute the threshold value τ. As soon as at least k objects have been seen whose grade is at least equal to τ , then go to step 5.
3. For every object x that was seen under sorted access in the step 2, do the random access to the other lists to find the grade $x_i = r_i(x)$ of object x in every list. Then compute the grade $S(x) = F(x_1, \ldots, x_m)$ of object x. If this grade is one of the k highest ones we have seen, then remember object x and its grade $S(x)$.
4. Second check: As soon as at least k objects have been seen whose grade is at least equal to τ , then go to step 5, otherwise go to step 1.
5. Let Y be a set containing the k objects that have been seen with the highest grades. The output is then the graded set $\{(x, S(x)) : x \in Y\}$.

The easiest heuristic is the heuristic in Threshold algorithm [7]. This heuristic chooses all the lists as candidates, i.e. $H(z)_i = z_i{+}1$ for every i. For overview of other heuristics see [9, 10]. This algorithm is correct [7] for any heuristic and instance optimal for some of heuristics [7, 10, 11]. The instance optimality guarantee that for any data the algorithm do at most m^2 times more accesses then in the ideal case.

4 Three phased no random access (3P-NRA) algorithm - uses only sorted access

3P-NRA algorithm was firstly presented by P.Gurský in [13] and it is an improvement of NRA algorithm [7].

First of all we need to define worst and best value. Given an object x and subset $V(x) = \{i_1, \ldots, i_n\} \subseteq \{1, \ldots, m\}$ of known attributes of x, with values $x_{i_1}, \ldots, x_{i_n}$ for these fields, define $W_V(x)$ (or shortly $W(x)$ if V is known from context) to be minimal (*worst*) value of the aggregation function F for the object x. Because we assume that F is monotone aggregation function, we can compute its value by substituting for each missing attribute $i \in \{1, \ldots, m\} \backslash S$ the value 0. For example if $V(x) = \{1, \ldots, g\}$ then $W_V(x) = F(x_1, \ldots, x_g, 0, \ldots, 0)$.

Analogously we can define maximal (*best*) value of the aggregation function F for object x as $B_V(x)$ (or shortly $B(x)$ if V is known from context). Since we know that values in the lists are ordered descended we can substitute for each missing property the values along the vector z. For example if $V(x) = \{1, \ldots, g\}$ then $B_V(x) = F(x_1, \ldots, x_g, u_{g+1}, \ldots, u_m)$.

The real value of the object x is $W(x) \leq S(x) \leq B(x)$. Note that the unseen object (no attribute values are known) has $B(x) = \tau = F(u_1, \ldots, u_m)$ and $W(x) = F(0, \ldots, 0)$. On the other hand if we know all the values $W(x) = B(x) = S(x) = F(x_1, \ldots, x_m)$.

The 3P-NRA algorithm is as follows:

I. Descending with the threshold and the heuristic $H1$
   0. Set z:=(0,. . . ,0)
   1. Set the heuristic H1 and do the sorted access in parallel to each of the sorted lists to all positions where $H1(z)_i = z_i{+}1$. Put $z_i = H1(z)_i$.

2. For every object x seen under sorted access in the step 1, compute $W(x)$ and $B(x)$. If the object x is relevant, put x in the list T, that is the list of relevant objects ordered by $worst$ value (an object x is relevant, if less than k objects was seen or $B(x)$ is grater than k-th biggest $worst$ value in T). If the object x is not relevant remove it from T.
3. If we have at least k objects in T with greater $worst$ value than τ go to phase II. otherwise go to step 1 of phase I.

II. Removing irrelevant objects
Compute *best* value for each object in T between the $(k+1)$-th and the last one. If an object is not relevant remove it from T. If $|T| = k$ return T otherwise go to phase III.

III. Descending with the heuristic $H2$
1. Set the heuristic $H2$ and do the sorted access in parallel to each of the sorted lists to all positions where $H2(z)_i = z_i+1$. Put $z_i = H2(z)_i$.
2. For every object x that was seen under sorted access in the step 1 of this phase do: If $x \notin T$ ignore it, otherwise compute $W(x)$ and $B(x)$. If the object x is relevant, move x to the right place in the list T. If the object x is not relevant remove it from T.
3. If $|T| = k$ return T
4. If by moving in T the k-th value of T was changed or the value of τ was decreased go to phase II, otherwise repeat phase III.

As heuristic H1 we can choose the heuristic from Threshold algorithm again. As heuristic H2 we can use heuristic *holes* [13], which chooses as candidates the lists with lowest number of known values in T. This algorithm is also correct [7] and instance optimal with the use of heuristic from Threshold algorithm.

5 Extensions

In all proposed extension we assume that the lists $L_1, \ldots, L_m$ are ordered by real values of properties from the smallest to the biggest, thus not from the best to the worst (it is not possible in general case). For example the distances from the centre of the city will be ordered from nearest to the most far. Next we assume that we have user fuzzy function for each property and it is one of 4 types like on figure 1. Let f_i be the fuzzy function for the list L_i. The overall fuzzy score of the object x will be $S_f(x) = F(f_1(x_1), \ldots, f_m(x_m))$. We will call $f_i(x_i)$ the fuzzy value of i-th property of the object x.

The main principle of both TA and 3P-NRA algorithms is to retrieve top k objects correctly without reading whole lists, thus using as low number of accesses as possible. Adding fuzzy functions, the situation is getting more complicated. Descending the lists by the real value causes that the threshold in previous algorithm does not guarantee the correctness any more. However the situation can be better when for some lists we have the fuzzy functions of type A. In this case such lists provide data from the best to the worst i.e. as it was in previous algorithms.

In the following we assume that lists $L_1, \ldots, L_a$ are all lists with fuzzy functions of type A, $L_{a+1}, \ldots, L_b$ are all lists with fuzzy functions of type B, and $L_{b+1}, \ldots, L_c$ and $L_{c+1}, \ldots, L_m$ are all lists with fuzzy functions of types C and D respectively.

5.1 Restricting sorted access

Bruno, Gravano, and Marian [14] discuss a scenario where it is not possible to access certain of the lists under sorted access. They did not consider fuzzy functions, but their solution can be correctly used in our case without any change. The only condition is that we have at least one list with the fuzzy function of type A, so we can do sorted access to this list. This solution is correct and instance optimal [7]. Algorithm is as follows.

1. Do sorted access in parallel to each list $L_1, \ldots, L_a$. For an object x seen under sorted access in some list, do random access as needed to the other lists to find the grade x_i of object x in every list L_i. Then compute the grade $S_f(x) = F(f_1(x_1), \ldots, f_m(x_m))$ of object x. If this grade is one of the k highest we have seen, then remember object x and its grade $S_f(x)$.
2. For each list L_i with $i \in \{a+1, \ldots, m\}$, let $u_i = 1$. As soon as at least k objects have been seen whose grade is at least equal to τ, then halt.
3. Let T be a set containing the k objects that have been seen with the highest grades. The output is then the graded set $\{(x, S_f(x)), x \in T\}$.

5.2 Reading whole list or waiting for a maximum

Now we propose the first solution for extension of 3P-NRA. We will read all the lists that have fuzzy functions of types B or D. Next we will read all the lists that have fuzzy functions of type C until they grow to the maximum fuzzy value. We can save accesses mainly to the lists with fuzzy function of type A and partially in the lists with type C. This solution can be helpful especially when we extend 3P-NRA algorithm. We will add a phase zero before the algorithm 3P-NRA:

0. Waiting for descending values
    0. For all i set $u_i = 1$ and compute the threshold value $\tau = F(u_1, \ldots, u_m) = F(1, \ldots, 1)$. In this phase u_i is fixed for all i because we need to keep the threshold to be the upper bound of all unseen object values.
    1. Choose one list L_i from $L_{a+1}, \ldots, L_b, L_{c+1}, \ldots, L_m$ and read whole list L_i by sorted accesses. If there is no such a list go to step 3. Put all objects to list T ordered by the *worst* value. Set $u_i = 0$ and compute new threshold. If any object seen is no more relevant (when its *best* value is smaller than *worst* value of the k-th object in T), remove it from T. After all, set $z_i = n$ i.e. to the last position in the list.
    2. If $|T| = k$ and τ is smaller or equal than the *worst* value of the k-th object in T, return T and halt. If there are unread lists with fuzzy function of type B or D and there are more than k relevant objects in T go to step 1. Otherwise go to step 3.

3. For each list L_i from $L_{b+1}, \ldots, L_c$ read the list L_i up to position where the fuzzy value of the property reach the maximum value i.e. value 1. If there is no such a list go to step 1 of phase I. After each list do the same check as in step 2.

It can be easily seen that adding the phase 0 before 3P-NRA solves our problem correctly. The main idea of this phase is to reach the best values in all lists after whose we have the same start situation as we had in the original 3P-NRA algorithm.

Theorem 1. *The last extension of the algorithm 3P-NRA is correct.*
Proof. The objects are removed only when they are not relevant. The question is: if an object become irrelevant, should it become relevant again? By other words if its *best* value is smaller than $worst$ value of the k-th object in T should its *best* value be later greater? Since u_i is fixed to 1 for all i for current read list (it does not change by sorted access), it is larger or equal to the real value of the object. Moreover we assume that the aggregation function is monotone. Hence the best value of any object decreases only. Since the list T is ordered by $worst$ values and $worst$ values of all objects increase only, the $worst$ value of the k-th object in T increases only. Considering both these facts, we can see that when the object become irrelevant it cannot become relevant anymore. The last thing to solve is the question if all possible objects are considered to be in top k. If at least one whole list is read in step 1 all objects are considered automatically. If we read the lists in phase 0 only in step 3 again all objects up to highest fuzzy value are read. The rest of values are read in other phases (I.-III.) and as it was shown in [13], these phases are correct. □

The phase 0 can also be put before algorithm TA. This algorithm works well, when we do not have the property with the fuzzy function of type A too.

5.3 Two ways descending

The next extension of both TA and 3P-NRA algorithms will cause the same performance in the case of each fuzzy function type as the algorithms TA and 3P-NRA in the original task. To reach such a performance, we need lightly upgrade the functionality of data sources. We will require:

- A source will provide two lists for sorted access - first will send objects with property values ordered from the biggest to the smallest (descending order) and second will send data from the smallest to the biggest (ascending order). It can be implemented for example as two pointers on the same ordered list - one goes from left to right and the second goes from right to left.
- Lists can start sending data from the specified value.

When we have such functionality we can easily simulate the source that sends data from the best to the worst. Moreover we guarantee that we do not need any reordering or any other computation on the side of source.

When we have a property with fuzzy function of type A or B we can easily simulate the "best-worst" source by choosing the suitable list of the source -

ascended or descended. In this case, one request for a sorted access from a central algorithm means to do one sorted access to the real source and computation of fuzzy value.

To simulate the source using the fuzzy function of type D, we can use both lists and start from the first record in each list. Thus we get the biggest value of the active domain of the given property from the first list and the smallest value of the same property from the second list. After the computation of fuzzy values of both retrieved values, we can send to the algorithm the greater one. After next request from the algorithm, we must do the sorted access to the list from which we sent the value last time and again compare fuzzy values computed from both lists.

Assume that from the top of the first list we will retrieve fuzzy (computed by fuzzy function) values $(o_1, 1.0), (o_2, 0.8), (o_3, 0.7), \ldots$ and from the second list fuzzy values $(o_4, 0.9), (o_5, 0.8), (o_6, 0.6), \ldots$. After first request we need to do sorted access to both lists and retrieve objects o_1 and o_4 with fuzzy values 1.0 and 0.9 respectively. 1.0 is greater than 0.9, so we send to the algorithm $(o_1, 1.0)$. After next request, we will do the sorted access to the first list and retrieve object o_2 with fuzzy value 0.8. We send greater $(o_4, 0.9)$, thus the next request will cause the sorted access to the second list. After receiving $(o_5, 0.8)$, we can randomly choose, which object has to be sent. If we choose o_2, the next sorted access will be to the first list. The objects o_5, o_3 and o_6 will be sent at the end.

The simulation of the source using the fuzzy function of type C needs also the second requirement - to start sending data from the specified value. If we want to send the values from the best to the worst, we need to start from the value with maximal fuzzy value. It means to start in the "middle" of the list to both ends, or also from the same specified value in both ordered lists. Now we are in the same situation as in the case of fuzzy function of type D and we can use the same combination procedure of two ordered lists.

As can be seen using this approach we can simulate the "best-worst" sources with the same number of accesses to the sources except one sorted access for each source with fuzzy function of type C or D. Thus we can use all known algorithms developed for the "best-worst" sources with the same good performace.

We use the two ways descending method in the tool $top - k$ *aggregator* in the project NAZOU[1]. The main task of this tool it to find top k job offers for a user.

6 Conclusion

In this paper we extended the model of distributed multifeature querying by adding user specification of preferences to properties values. Such a model allows better specification of the idea of good object using object properties. We propose the extensions of known algorithms to work over this model. Proposed solutions are needed especially in the cases when we cannot reorder the lists in provided

[1] http://nazou.atrip.sk

sources. Reordering is quite difficult when fuzzy functions come together with the query.

In the future work is the comparison of proposed algorithms over real data. In present we have implementation of the last extension. We can see from the design of the algorithms that it is the best, because it works as good as current well known algorithms over simplest model. Other algorithms should be useful, when there cannot be required functionality in the sources. On the other side the extensions work with individual sources, hence the approaches should be combined.

References

1. U.Güntzer, W.Balke, W.Kiessling *Optimizing Multi-Feature Queries for Image Databases*, proceedings of the 26th VLDB Conference, Cairo, Egypt, 2000
2. U.Güntzer, W.Balke, W.Kiessling *Towards Efficient Multi-Feature Queries in Heterogeneous Environments*, proceedings of the IEEE International Conference on Information Technology: Coding and Computing (ITCC 2001), Las Vegas, USA, 2001
3. W.Balke, U.Güntzer, J. Zheng *Efficient Distributed Skylining for Web Information Systems*, proceedings of the 9th International Conference on Extending Database Technology (EDBT 2004), LNCS 2992, Heraklion, Crete, Greece, Springer, 2004
4. W.Balke, U.Güntzer *Multi-objective Query Processing for Database Systems*, proceedings of the 30th International Conference on Very Large Databases (VLDB 2004), Toronto, Canada, 2004
5. W.Balke, U.Güntzer *Efficient Skyline Queries under Weak Pareto Dominance*, proceedings of the IJCAI-05 Multidisciplinary Workshop on Advances in Preference Handling (PREFERENCE 2005), Edinburgh, UK, 2005
6. R.Fagin *Combining fuzzy information from multiple systems*, J. Comput. System Sci., 58:83-99, 1999
7. R.Fagin, A.Lotem, M.Naor *Optimal Aggregation Algorithms for Middleware*, proc. 20th ACM Symposium on Principles of Database Systems, pages 102-113, 2001
8. S.Börzsönyi, D.Kossmann, K.Stocker *The Skyline Operator*, ICDE 2001: 421-430, Heidelberg, Germany, 2001
9. M.Vomlelová, P.Vojtáš *Pravděpodobnostní pohled na víceatributové dotazy v distribuovaných systémech*, Proceedings of ITAT 2005, p. 167-175, 2005
10. P.Gurský, R.Lencses *Aspects of integration of ranked distributed data*, proc. Datakon , ISBN 80-210-3516-1, pages 221-230, 2004
11. P.Gurský, R.Lencses, P.Vojtáš *Algorithms for user dependent integration of ranked distributed information*, technical report, 2004
12. P.Gurský, T.Horváth *Dynamic search of relevant information*, Proceedings of Znalosti 2005, pages 194-201, 2005
13. P.Gurský *Algoritmy na vyhľadávanie najlepších k objektov bez priameho prístupu*, Proceedings of Znalosti 2006, pages 95-105, 2006
14. N. Bruno, L. Gravano, and A. Marian *Evaluating top-k queries over web-accessible databases*, proceedings of the 18th International Conference on Data Engineering. IEEE Computer Society, 2002.
15. Šuc, D. *Machine Reconstruction of Human Control Strategies*, Volume 99 of Frontiers in Artificial Intelligence and Applications. Amsterdam, IOS Press, 2003.

Viewing FOAF – Development of a Metadata Explorer

Josef Petrák

Faculty of Informatics and Statistics, University of Economics, Prague
Winston Churchill Sq. 4, 130 67, Praha 3, Czech Republic
jspetrak@gmail.com

Abstract. Social networks are widely accepted application of Semantic Web technologies and are also interesting for general public. Nowadays there is a lack of quality user-friendly browser which could express meaning of the metadata stored in the most of FOAF profiles and present this knowledge in human-understandable form. The aim of the article is to inform about development of the AFE (Advanced FOAF Explorer) which is intended to perform these services and supersede one similar project.

Keywords: social networks, Semantic Web, RDF, FOAF, PHP5

1 Introduction

Friend Of A Friend vocabulary (FOAF) [1], an ontology used for description of personal profiles and relations among people, is well-known and popular in the Semantic Web [2] community. Thank to the user-friendly tools such as FOAF-a-Matic [5] people who are not familiar with RDF [14] can create their own profiles and publish them on the Internet. This is a good because may start spreading of the Semantic Web technologies to the public.

But if these users create such profiles, they also require having a chance to view their profiles or to browse the profiles of other people. Nowadays there are several "viewers" but their features are limited. These restrictions do not allow to wide spreading of FOAF users. These limitations are discussed in section 2.

The aim of the project "Advanced FOAF Explorer" [9] is to develop an FOAF explorer with user-friendly XHTML [17] output. To allow the developers to easily extend the code and add support for new vocabularies which may extend existing ontology. The most important task is the show relations among various resources because this is the main positive of using FOAF.

I already developed a first beta version based on PHP. It shows randomly chosen subset of FOAF terms and relations defined in the ontology extension called "relationship" [13]. The structure of the output shows how should be the data presented. History of this version and details about implementation are discussed in the section 3. Further development should produce new version of this application. The basic ideas and implementation details are described in the section 4.

1.1 Brief introduction into social networks

If we consider definition of social networks published on the Wikipedia [3], we have to restrict the term to on-line social networks. It is an on-line community based on a Website which allows each member to communicate, make friends among other members, to discuss topics which are interesting to this community … FOAF allows to store information about member of any community, the relationships among them (and type of the relation such as friend of someone, parent of someone, enemy of someone, roommates with someone …). This is interesting for most of the people because making contacts, making friends and entertainment are the basic needs of almost everyone.

1.2 What is FOAF?

The basic explanation is that FOAF is ontology. It is defined using RDF Schema [18] and OWL [15]. If you see the specification [13], you can find there a lot of classes and their properties. Using them you can create various RDF[1] statements about you, your relatives, friend and the people who you know. You can describe your work, schools, interests, your Websites and if you use some additional modules, you can also describe countries you have already visited, languages which you reads, speaks or writes and many other more or less interesting information.

Let's have a look at the basic structure of a FOAF profile. In the Figure 1. I gave an example in XML [16] syntax of RDF. It contains some information about a person, and statement that this person knows somebody else.

As you can see, the FOAF defines terms for commonly required properties such as name, mailbox, and gender. But the specification defines many more of them such as chat IDs for instant messaging services (AIM, ICQ, Jabber, Yahoo!, and MSN), it allows to state, that any person has a homepage, a weblog, to link depiction, personal interests. The most interesting property is `<foaf:knows>` which states that a person knows any other person. The type of the relation is not defined. If you want to explicitly define the quality of the relationship between to persons, you have to use extension RELATIONSHIP [13] which contains a lot of interesting sub-properties, e.g. previously mentioned friend of, enemy, of, roommates with and many other such as works with, employees, knows by reputations, sibling of, loves, … The complete view about the vocabulary you can make reviewing the specification.

1.3 How to display it on the Website

There are some technical limitations which do not allow to directly put the FOAF profile into the code of any Website. You can store your profile in the file and link it to the page using element `<link>` with parameter `rel="meta"`.

[1] More information about *Resource Description Framework* and also the specification you can find at http://w3.org/RDF

```
<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF
  xmlns:foaf="http://xmlns.com/foaf/0.1/"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#" >
  <foaf:Person rdf:ID="adam-smith">
    <foaf:name>Adam Smith</foaf:name>
    <foaf:gender>male</foaf:gender>
    <foaf:mbox rdf:resource="mailto:adam@smith.name" />

    <foaf:knows>
      <foaf:Person>
        <foaf:name>John Doe</foaf:name>
        <foaf:knows rdf:resource="#adam-smith"/>
      </foaf:Person>
    </foaf:knows>
  </foaf:Person>
</rdf:RDF>
```

Fig. 1. The example of basic structure of FOAF profiles.

2 Motivation for development

To have a complete view to the history we have to explain why the development of this explorer started was, and why it is called to be “advanced”. In the years in which the FOAF itself was created there was a need for quality and extendible browser which could serve information stored in such profiles to the user in XHTML format. There are several applications used for parsing in viewing these profiles, e.g. FOAF Web View [7] or Plink.org [8] which also integrated a simple storage for viewed metadata. But they have various problems. First of all they do not interpret all or at least most of the FOAF terms. If an application interprets all terms, its output does not have a nice look or the information are presented in illogical order so the user is confused and cannot find what he is looking for. They often do not show relations among various resources (people, people and projects ...). Another problem is that FOAF allows easy extendibility and there are many widely used 3rd party extensions. Usually these browsers do not support these extensions. And the last problems is that the interface does not explicitly explain the meaning of the terms and do not allow internationalization so the non-English speak users cannot use them.

Nowadays the most used application is FOAF Explorer [6] developed by Morten Frederiksen. It is based on XSL Transformations. It supports various extensions, displays some relations among the resources. But the look of the output could be created better, show more information in a user-friendlier way. And this is chance for the project of AFE!

3 History of the project

The development was started by two students – Josef Petrák and Michal Trna - in the summer 2004. The very first goal was to quickly create first beta-version for viewing of authors' profiles. Further development should have focused on cleaning the application code and to implements most commonly used FOAF extensions. Unhappily its development was frozen in the same year. This beta version is still available [9] and the community was informed about it on the IRC channel [4].

3.1 The first beta – details of the implementation

Parsing of the RDF files is done using the library RDF API for PHP [11] which is broadly accepted library for manipulation with RDF files in the PHP applications. The scenario of producing result is easy.

1. User writes URL of any profile which he wants to browse into the form.
2. Application tries to download the FOAF file. If it is successful, it parses the RDF and generates in memory a RDF graph.
3. To simplify later generating the output, the application transforms a special structure from the graph. In fact the structure is an 3D associative array where the indexes represents:

   – URIref of the resource
   – URIref of the property describing the resource
   – Auto incremented integer identifying the values of the property.

   The algorithm of creating the array structure is shown in the Figure 2.

After that, huge function `format_person($data, $model, $id_person)` goes through this array and for each resource generates XHTML code which will appear in the output page. It also generates anchors which allow the user to click on it and see the relations between the resources (on the figure 3 there is shown simple output from an existing profile). You can imagine that this solution is quite "dirty", it lacks any design patterns but it was chosen for quick implementation of the problem. And as we can see, it works well.

```
function get_data_structure ($model) {
  $res = array();

  for ($iter = $model->getStatementIterator();
                                $iter->hasNext();) {
    $statement = $iter->next();
    $predicate = $statement->getLabelPredicate();
    if (!is_array($res)) {
      if (!is_array($res[$predicate])) {
        $res[$predicate] = array();
      }
    }
    $res[$predicate][] = $statement->getLabelObject();
  }
  return $res;
}
```

Fig. 2. The algoritm for generating array structure from a RDF graph.

Now is time to change the implementation, clean-up the design and remake the output. Future plans are all described in the section 4.

4 Future plans

4.1 MVC Design Pattern

In any well-designed application it is necessary to separate data model, view layer and logic which loads and manipulates with the data. In modern programming languages such as Java or .NET there various frameworks, which allows simple use of the pattern of **Model – View – Controller**. We can mention *Struts*[2] and *JavaServer Faces* in Java, package *Web.Forms* in .NET. For PHP there are only limited amount of such frameworks. The most advanced is component-driven framework called PRADO[3], but it is too complex to be easily implemented. Better way is to implement own simple MVC solution. The new library is based on PHP5 object model. All important components are defined using interfaces andd errors are handled using exceptions.

[2] http://jakarta.apache.org/struts/
[3] http://www.xics.com/

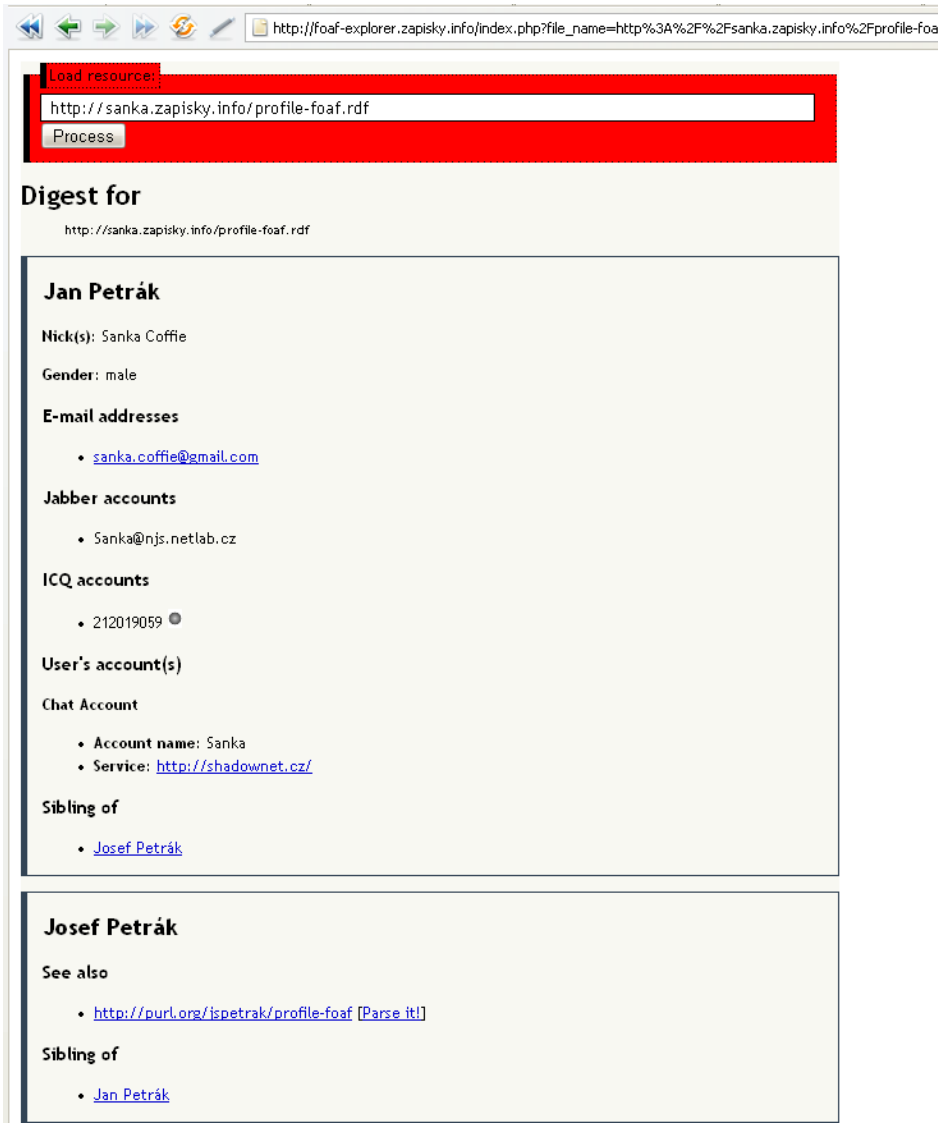

Fig. 3. An example of output from the first version of Advanced FOAF Explorer. The address of the profile is shown in the address bar. Clicking on the link "Parse it!" we can show other linked profiles in the viewer.

Each application is started using static method, such as `AfeApplication::run();`, the application will recognize name of template from the PHP file name. For `index.php` is assigned the template

`views/_index.php`. The templates are written also in PHP using `foreach` construction and using getter methods of all objects from data model.

The application class manages all operations. It starts loading resources; stores got data in any model object and the calls a viewer which uses given template and according to the instruction views the information. The core class also catches any thrown exception and accesses it by a getter method. Using getter and setter methods is the basic concept of the design of this class – it makes the API clearer.

There is a class which parses all information from server configuration variables, request variables (information sent by GET or POST methods) as well. Further development of this MVC framework will introduce common configuration files in INI or XML[4] format.

4.2 History of viewed profiles

To control viewed files and have a chance, how to explore unknown extensions of FOAF, it is necessary to store the data about browsed files. To maximally simplify the task, it is designed simple database table containing three columns: URI of the viewed file, date and time when the request was received from a user and count of triples stored in the profile. Data may be store in any relation database but the best for simple use in PHP5 are integrated database SQLite or MySQL 4.1 Both of them offers object-oriented approach to the API (through the `SQLiteDatabase`, or `mysqli` object respectively). The structure of the database table is shown in figure 3.

```
CREATE TABLE ‘history’ (
  ‘uri’ VARCHAR(255) NOT NULL,
  ‘dt’ DATETIME NOT NULL,
  ‘triples’ INT UNSIGNED NOT NULL,
  PRIMARY KEY ( ‘uri’ , ‘dt’ )
);
```

Fig. 4. Structure of SQL table for storing the history

To create fully persistent approach, there is a class `History` in the application with methods for loading and saving the data. If it is necessary to change the database, the implementation will be changed only in this class and the rest of the application will be not influenced by this change. The question is if implement this class straight or define an interface and implement classes using drivers for all considered database drivers (which were mentioned previously).

[4] eXtensible Markup Language

To complete the persistence, there is the object `HistoryItem` which represents one row from the table. It defines getter and setter methods for manipulation with the data in the object.

4.3 Data model

The biggest change from the first version is the data model. Due to ineffective work with the solution of an associative array it was left and now fully object – oriented solution is going to be created. The basic concept is to represent all known RDF resources as classes and their properties as getter methods which return array with values of property of the same type. We can also describe the hierarchy of the resource, e.g. that `<foaf:Person>` is a subclass of `<foaf:Agent>` (using keyword `extends`). It means, that Person may have all of properties, which are defined for the Agent and it also may have some new properties which the Agent does not have. It also allows easily extend the application model. Changing the properties or creating new classes is a question of seconds.

In the Java, the class in the hierarchy of classes is `java.lang.Object`. Also our data model has a super class which defines common methods and properties. They are common for all types of resources. This class represents the resource `<rdf:Resource>`. There is also interface `Labelable` which defines one common methods called `getCommonLabel()`. This method offers label which will appear in the name resource (such as headers or image titles). The code of this interface and the super class `RdfResource` is shown in the figure 5.

As you can see, the method `__toString()` is used for testing purposes. It returns the dump of structure of the class. The most interesting method is `parseRequiredProperty()` it has to be called in the constructor of the model classes for every property which we intend to read from the RDF graph. Note down that it is necessary that if we declare any subclass, we have to send the parameters to the super class otherwise it will not be loaded values of properties defined in the parent classes.

```
interface Labelable {
  public function getCommonLabel();
}
class RdfResource implements Labelable {
  private $uriRef;
  private $rdfType;
  private $foafName;
  public function __construct(&$model, &$resource) {
    $this->uriRef = $resource;
    $this->parseRequiredProperty($model, $this->rdfType,
RDF::TYPE());
    $this->parseRequiredProperty($model, $this->foafName,
FOAF::NAME());
  }
  protected function parseRequiredProperty(&$model,
&$objectProperty, &$property) {
    for ($i = $model->find($this->uriRef, $property, NULL)->
getStatementIterator(); $i->hasNext();) {
      $stmt = $i->next();
      if (is_null($objectProperty)) $objectProperty =
array();
      $objectProperty[] = $stmt->getLabelObject();
    }
  }
  public function getRdfType() { return $this->rdfType; }
  public function getFoafName() { return $this->foafName; }
  public function __toString() { return print_r($this, true);
}
  public function getCommonLabel() {
    if (count($this->foafName)>0) return explode(', ', $this
->foafName);
    else return $this->uriRef->getUri();
  }
}
```

Fig. 5. The code of the `RdfResource` class and the interface `Labelable`

4.4 Internationalization

Internationalization allows non-English speaking users to use this application. We have to separate labels from the index and output page, store them in one place and create multiple translations. If we consider existing solutions for internationalizations, there is mechanism represented by function `gettext`[5]. It is standard for PHP application so it is not necessary to create any other now framework.

There is also one important problem. We have to find volunteers who can translate these texts into languages different from Czech or English. In sure, that if we announce new version on the FOAF IRC [4] channel and inform about possibility to translate the user-interface, we will find the volunteers who can help us to translate the UI into their mother language.

4.5 Data-binding

We considered implementing a mechanics of data-binding which could automatically generate data model classes from given RDF Schemas and ontology. Even if it is a good idea, we did not find any existing tool for this purpose. Our project is not targeted to create any kind of this application. But it could be useful for developers from the Semantic Web community. There is space for other developers to implement it …

One problem of this solution could be that the definition of FOAF ontology and its modules are stored in different files. Some of them are Ontologies in OWL and some of them only RDF Schemas. So this tool should generalize the generate model and ignore some constructions which are not common for RDFS and OWL (and its versions).

5 Conclusion

It is necessary to define all necessary objects in the data model and to consider which extension support – the most important extension is the module RELATIONSHIP, which extends the basic concept of making relations among people defining a lot of new sub-properties. After that there should not be any problem which could slow down the development. The most important goal of this project is to show that building of any Semantic Web application is not so hard as many of developers think.

After releasing the tool and announcing it to the international FOAF community we expect big interest in the source codes, details about the implementations and volunteers who can create multiple translations. The aim of the project is to pay attention to the Semantic Web, especially FOAF and to promote its using among ordinary (understand non-programmers) users. After finishing all of these tasks, we

[5] Documentation of gettext: http://php.net/gettext

can focus on improving our MVC framework, to the development of other RDF tools which may help our author easier build their applications based on RDF graphs.

References

1. Project *Friend of a Friend*: http://foaf-project.org/
2. Wikipedia. Term *Social Networks*: http://en.wikipedia.org/wiki/Social_network
3. Wikipedia. Term *the Semantic Web*: http://en.wikipedia.org/wiki/Semantic_Web
4. *IRC channel of the FOAF developers*: Server irc:irc.freenode.net, channel *#foaf*
5. Application FOAF-a-Matic: http://www.ldodds.com/blog/archives/000087.html
6. Application *FOAF Explorer*: http://xml.mfd-consult.dk/foaf/explorer/
7. Application *FOAF Web View*: http://eikeon.com/foaf
8. Application *People Link.org*: http://beta.plink.org/, the project was shut down.
9. Application *Advanced FOAF Explorer* (AFE): http://foaf-explorer.zapisky.info/
10. Article "*Jednoduchý MVC framework napsaný v PHP*" (Simple MVC framework written in PHP) posted by Josef Petrák on 8th February 2006: http://zapisky.info/?item=jednoduchy-mvc-framework-napsany-v-php
11. Library *PHP API for PHP*: http://www.wiwiss.fu-berlin.de/suhl/bizer/rdfapi/
12. Specification of *the FOAF Vocabulary*: http://xmlns.com/foaf/0.1/
13. RDF Schema *RELATIONSHIP: A vocabulary for describing relationships between people*: http://vocab.org/relationship/.
14. RDF Primer, Frank Manola, Eric Miller, W3C Recommendation from 10th February 2004, The latest version is available at http://www.w3.org/TR/rdf-primer/.
15. OWL Web Ontology Language Reference, Dean M., Schreiber G (Editors); van Harmelen F., Hendler J., Horrocks I., McGuinness D.L., Patel-Schneider P.F., Stein L.A. (Authors), W3C Recommendation, 10 February 2004. The latest version is http://www.w3.org/TR/owl-ref/.
16. Extensible Markup Language (XML) 1.0, Second Edition, Bray T., Paoli J., Sperberg-McQueen C.M., Maler E. (Editors), World Wide Web Consortium, 6 October 2000. The latest version is http://www.w3.org/TR/REC-xml.
17. XHTML 1.0 The Extensible HyperText Markup Language (Second Edition), World Wide Web Consortium. 26 January 2000, revised 1 August 2002. The latest version of XHTML 1 is available at http://www.w3.org/TR/xhtml1/.
18. RDF Vocabulary Description Language 1.0: RDF Schema, Brickley D., Guha R.V. (Editors), W3C Recommendation, 10 February 2004. The latest version is http://www.w3.org/TR/rdf-schema/.

Using WordNet Glosses to Refine Google Queries

Jan Nemrava

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

Abstract. This paper describes one of the ways how to overcome some of the major limitations of current fulltext search engines. It deals with synonymy of the web search engine results by clustering them into relevant synonym category of given word. It employs WordNet lexical database and several linguistic approaches to classify results in search engine result page (SERP) in appropriate synonym category according to WordNet synsets. Some methods to refine the classification are proposed and some initial experiments and results are described and discussed.

Keywords: text mining, text classification, web search engine, WordNet gloss

1 Introduction

Fulltext search engines have recently become a basic tool for acquiring arbitrary information from the World Wide Web. The amount of queries inserted into Google rises rapidly and so does the number of indexed pages. *'To Google'* became a commonly used verb describing the act of searching any information on the Internet. Nowadays, Google has an Internet domain in 135 world countries and with its 88 language interfaces is a world most leading search engine. This determines to use Google and other search engines as a most suitable tool for an easy access to any kind of information from our desktop PC and makes the proclaimed information society viable. Nevertheless, still there exist some limitations that play an important role in searching information within a keyword based search interfaces. One of the keyword-based web search major problems is that people tend to insert too general queries (according to Search Engine Journal [1], in 2004 more than 50% of all queries inserted were one or two words long), which leads to huge amount of returned hits to a given query. The way how to deal with a huge amount of returned web pages is to arrange the results according to their proper meaning using their synonyms or the word sense disambiguation. The purpose of this paper is to describe some techniques how to arrange returned web sites into appropriate synonym classes using large lexical database *WordNet*[1] for discovering the synonyms and *Hearst Patterns* for discovering is-a relations between the queried term and its possible superclass (i.e. hypernym) concept.

[1] http://wordnet.princeton.edu/

The structure of this paper is as follows: Section 2 describes our motivation, section 3 contains description of all information sources that were used. Our goals and techniques used for this approach according with a given examples, some drawbacks and limitations are discussed in section 4. Before concluding, section 5 discusses some relevant work on this topic.

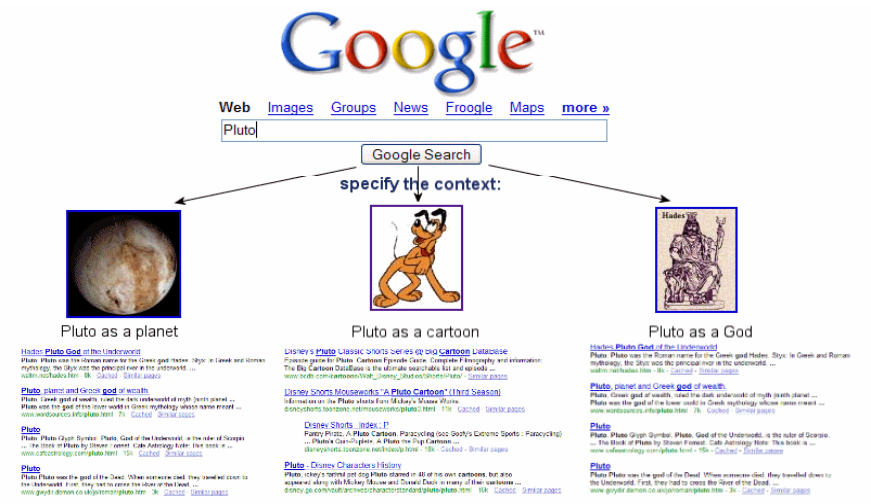

Fig. 1. A context suggestion interface

2 Motivation

As it was stated in the Introduction, the problem of ambiguous queries presents a strong limitation of current web search technology. There are already emerging some query refinement techniques, which allow users to zoom into more specific query, but most of the time they only provide a "query modification" lists as a single list without distinguishing between the real meanings of given word (e.g. Ask Jeeves[2]). Another query refinement method recently introduced by leading fulltext search engine is offering real time suggestions while the user is typing in his query. One of the advantages is that the user sees the most suitable word form for a particular search in the realtime (though the suggested word may not be the grammatically or semantically best one, but it is the one that is

[2] http://www.ask.com

used by the most of the users). *Google Suggest*[3] is good example of this method. To our knowledge there isn't any fulltext search engine that would be able to separate returned results according to their meanings. Some efforts can be seen in *Vivisimo*[4], but is not known in public.
In this paper we would like to present approach that use existing dictionary and glosses describing its concepts together with the largest text corpora available, the Internet, to discover meanings that the word inserted can carry. This work was inspired by Philipp Cimiano's work on Pankow [4] system and the idea of using heterogenous evidence for confirming *is-a* relation.

3 Information Sources

In this section, we will describe the above mentioned techniques in detail. All approaches used here are well known among the Semantic Web [2] community for a long time. They are frequently used for ontology learning and creating is-a relations and taxonomies. Namely they are:

- **WordNet** - large lexical database containing words ordered in synsets (synonym sets).
- **Hearst Patterns** - technique exploiting certain lexico-syntactic patterns to discover is-a relations between two given concepts.
- **monothetic clustering** - information retrieval technique used for grouping documents according to specified feature.
- **fulltext search engine** - Google^{TM} API interface.
- **NLP** - natural language processing techniques.

3.1 WordNet

The main source of information is WordNet [7]. WordNet is a huge lexical database containing about 150,000 words organized in over 115,000 synsets for a total of 203,000 word-sense pair. Each word comes along with a short description called a *gloss*. The glosses are usually one or two sentences long. Beside the fact that all ordinary part of speech are present it contains nouns which are of major importance for us, because one of them is most likely a super concept (a hypernym) to the given word. This is a key idea of this paper.
After a user inserts some proper noun, it is looked up in a WordNet and all its meanings saved in WordNet are extracted together with their glosses. Each synonym contains just one gloss. Each gloss is preprocessed and then labeled by POS tagger. The preprocessing contains elimination of punctuation, hyphenation and stop words. Next step is POS tagging and only nouns are kept and saved as *candidate nouns*. Candidate nouns are words that can be potentially selected as a hypernym for a given term.

[3] http://www.google.com/webhp?complete=1
[4] http://www.vivisimo.com

3.2 Hearst Patterns

Hearst patterns are lexico-syntactic patterns firstly used by M.A.Hearst[8] in 1992. These patterns indicate the existence of class/subclass relation in unstructured data source, e.g. web pages. Examples of lexico-syntactic patterns that were described in [8] are following:

- NP_0 *such as* NP_1, NP_2,...,NP_{n-1} *(and | or)* NP_n
- *such* NP_0 *as* NP_1, NP_2,...,NP_{n-1} *(and | or)* NP_n
- NP_1, NP_2,...,NP_{n-1} *(and | or) other* NP_0
- NP_0 *(including—especially)* NP_1, NP_2,...,NP_{n-1} *(and | or)* NP_n
- and very common "NP_i *is a* NP_0"

Hearst firstly noticed that from patterns above we can derive that for all NP_i, $1 \leq i \leq n, hyponym(NP_i, NP_0)$. Given two term t_1 and t_2 we are able to record how many times some of these patterns indicate an $is-a$-relation between given t_1 and t_2. Some normalizing techniques should be employed as some of the patterns will likely occur more frequently than the others. Although Cimiano [3] noticed that Hearst patterns occur relatively rarely in closed corpus and as described later, it is applicable also on Internet, their results provide valuable information. The main drawback is that Google search does not offer to use proximity operators and with the query requested as an exact match user must enter exact order of the whole pattern. For example searching for pattern *"planets such as Pluto, Neptune and Uranus"* will provide about 50 results, while *"planets such as Pluto, Uranus and Neptune"* won't return any. The most powerful pattern that we use for primary decisions is the "NP_i *is a* NP_0".

3.3 Clustering

Associating documents to relevant category (synonym category in our case) is a task very similar to a classic information retrieval task named by van Rijsbergen[16] *polythetic clustering*, where documents' membership to a cluster is based on sufficient fraction of the terms that define the cluster. As stated in [17] creating is-a relations is a special case of polythetic clustering where subclass belongs only to one superclass and this means that the membership is based only on one feature, called *monothetic clusters*.
This alternative form of clustering has two advantages over the polythetic variety. The first is the relative ease with which one can understand the topic covered by each cluster. The second advantage of monothetic clusters is that one can guarantee that a document within a cluster will be about that clusters topic. None of this would be possible with polythetic clusters.

3.4 Google API

The world leading fulltext search engine provides direct access to its huge databases through Google API[5]. It has limited daily number of queries and compared to

[5] http://www.google.com/apis

HTML based interface it is relatively slow, but it provides easy access from any programming language. Each query is responded in the same way as is the HTML interface. User can get number of results, web page titles, links and snippets (short description of web page based either on META tag description or part of text with emphasized keywords). Our algorithm search for very specific text patterns and we are interested only in aggregate number of results.

Next session describes application of above described information sources and some initial results.

4 Discovering the synonym classes

It was already described in a section about WordNet, that certain nouns from so called glosses are of our main interest. According to our observation glosses mostly contain one noun that is a hypernym to the given concept. This is a core prerequisite for our method as our aim is to find that hypernym noun among the words in gloss. After some simple NLP methods are applied, we retrieve *candidate nouns* for each gloss. What follows is a description of concrete situation that our script has to deal with. The example is a term *Pluto* which can be found in three different contexts according to WordNet. Pluto can be either a planet, a god or a cartoon.

- **WordNet glosses** for concept Pluto
  - SYN 1 *a small planet and the farthest known planet from the sun; has the most elliptical orbit of all the planets*
  - SYN 2 *(Greek mythology) the god of the underworld in ancient mythology; brother of Zeus and husband of Persephone*
  - SYN 3 *a cartoon character created by Walt Disney*
- **Candidate nouns** for concept Pluto.
  - SYN 1 *planet;sun;orbit;planets;*
  - SYN 2 *Greek;god;underworld;mythology;brother;Zeus;husband;Persephone;*
  - SYN 3 *cartoon;character;Walt;Disney;*
- **Patterns applied** on SYN 1 - number of returned results is in brackets
  - "Pluto is a planet" (1550), "Pluto is planet" (145)
  - "Pluto is a sun" (2), "Pluto is sun" (0)
  - "Pluto is a orbit" (0), "Pluto is orbit" (1)
  - "Pluto is a planets" (0), "Pluto is planets" (0)

It is necessary to take into a consideration the total amount of web pages where the words are mentioned and use this value to normalize the values.

$$w(i) = tf(i)/TC(i) \tag{1}$$

where i represents the $i-$th synonym class, tf is number of results for given pattern and TC is number of web pages returned when querying two terms without any constraints, it represents the popularity of the given pair of terms.

Candidate for the hypernym noun is then simply the highest value from all synonymic class array.

$$W = max(w(i)) \tag{2}$$

This candidate noun needs to be validated and confirmed by another Hearst patterns. The problem with a necessity of strict word order was mentioned in previous session. We must cope with this problem in order to find another pattern to validate the results from "is a" step. Pattern NP_{n-1} *and other* NP_0 was chosen, because we predict its bias with strict word order to be the lowest among all remaining patterns. In this pattern we had to deal with creating a plural form of each *candidate noun*. Some simple rules were adopted, such as adding "ies" suffix at the end of the word when the last character is "y" etc.. No language exceptions were taken into consideration.

- Patterns tested in a **validation step** (returned hits are in brackets)
  - "Pluto and other planets" (57)
  - "Pluto and other planet" (0)
  - "Pluto and other suns" (0)
  - "Pluto and other sun" (0)
  - "Pluto and other orbits" (0)
  - "Pluto and other orbit" (0)
  - "Pluto and other planetss" (0)
  - "Pluto and other planets" (57)

Maximum value from the array is considered as *hypernym noun*. If both patterns determine the same noun, it is considered as a hypernym noun. In the opposite case some other techniques to confirm or reject this hypothesis should be applied. The possibilities are discussed in last section. The process of searching for the right hypernym noun is repeated for all synonym classes that were given by WordNet. Next paragraph discusses some results that were gained on a test set.

The test set consisted of about 50 of proper nouns from space, travel and zodiac area. At the beginning it was necessary to manually check whether all the words from the test set are listed in WordNet. The result was that 96% (i.e. 48 from 50) proper nouns have their gloss in WordNet. Then the above described script has been run on each of 50 test words. After all the tests has been carried out, it was necessary to check the correspondence of the discovered hypernym with the real world concepts.

We discovered, that from the test set, 62% (31 words which contained 61 synonymic classes in total) were assigned with a hypernym *correctly* and they corresponded to real life objects. 9 words and all their meanings were assigned *wrongly*. The remaining 16% contained mistake in assigning some of the synonym class. More detailed analysis of words that were incorrectly labeled can be found in Table 2.

Mining for other synonyms than those explicitly stated in WordNet would definitely provide better results in some cases, on the other hand the certainty of wrongly assigned hypernym noun would undoubtly rise.

Table 1. Overall precision

Total number of words in list	50	(100%)
Words listed in WordNet	48	(96%)
Correct	39	(78%)
- completely correct	31	(62%)
- partially correct	8	(16%)
Wrong	9	(18%)

Table 2. Statistics of wrongly discovered terms

Number of wrong instances	17	(100%)
Both patterns wrong	7	(41%)
"is a" correct, "and other" wrong	4	(23%)
"is a" wrong, "and other" correct	6	(35%)

Table 3. Examples of negatively labeled synonyms.

Proper Noun	"Is a" pattern	"and other" pattern
Greenland	island	Arctic
Reykjavik	Iceland	Iceland
Kenya	Great	Great
Luxembourg	-	-
Luxembourg	city	city

4.1 Results

We tested a set of 50 proper nouns from several different areas such as astronomy and zodiac. Some of these were chosen because they were tested with the abovementioned PANKOW system. From these 50 test concepts with 92 synonyms in total, we got precision 62 percent. The results were appropriate to estimations and with regard to the fact, that this technique has been recently implemented and is far from mature, we found them satisfying. There are several drawbacks and suggestion for future work that will be discussed in this section and in the conclusion.
One of the drawbacks is the system speed which depends on Google API responses which are quite slow recently. The average time to resolve one synonymic class is about 50 seconds with average 20 Google queries per one synonym class. Another objective drawback is the limitation of current Google web search interface. It has no proximity operators and the query must be either inserted as an exact match or connected with AND boolean operator. Besides these technological problems there is also a limited amount of daily queries to one thousand which is sufficient only to process about two tens of concepts, which currently presents the main obstacle.

5 Related work

This section discusses work related to exploitation of WordNet glosses to use them with query refinements. Since word ambiguity presents an important issue in Information Retrieval community, there has been a lot of efforts invested to discover how to deal with the problem. The importance of disambiguated words and concept further increased with introduction of ontologies as a core of the so called Semantic Web. Nowadays, there is an enormous effort on this research field. The most successful approaches so far, either reuse some knowledge stored existing sources (exploiting Web directories structure [9], dictionaries or tagged corpuses) or make use of the inherited redundancy of information that are present on Internet (e.g. Armadillo [5] or KnowItAll [6]). Both of these systems continually and automatically expands the initial given lexicon by learning to recognize regularities in the large repositories, either internal regularity to a single document or external across set of documents.
Query refinement based on a concept hierarchies was discussed in for example in [12] or by Kruschwitz in [10]. Project that also use similar ideas to ours is one called WordNet::Similarity [13]. It is a tool kit written in Perl implementing several algorithms for measuring semantic similarity and relatedness between WordNet concepts. Two of algorithms (lesk and vector measures in concrete) uses WordNet glosses. Lesk finds overlaps between two given glosses to count the relatedness of them. The vector measure creates a cooccurrence matrix for each word used in the WordNet glosses from a given corpus, and then represents each gloss/concept with a vector that is the average of these cooccurrence vectors.
Project that inspired this work is called PANKOW (Pattern-based Annotation

through Knowledge on the Web) and was created by Cimiano et al. [4]. This work focuses on application of Hearst patterns over a given ontology to discover is-a relations solely from Internet. Some of the data tested in our paper were actually taken from their work.

6 Conclusions

In this paper we presented an approach for discovering synonym classes of given proper nouns. We used some freely accessible information sources and connected them together to get new features for discovering meanings of given proper noun. List of some commonly used proper nouns was collected and the proposed method was tested with this list. From 50 test concepts with 92 synonyms in total, we got precision 62 percent.
It remains for further work to find out how to exploit the WordNet hierarchy and involve glosses from class instances and subconcepts. Introducing another validation pattern would definitely increase the precision of the system. So far, the system can handle only single word queries. Handling more words queries and deriving proper synonyms categories could be an interesting challenge. Another task would be to implement a way how to deal with words and concepts not included in WordNet. Cimiano's PANKOW similar system might be beneficial for this task.
Although this application has certain drawbacks, we showed that the idea of exploiting WordNet glosses for discovering certain facts about given concepts is viable and with some improvements in speed and precision it could serve as a helpful tool for unexperienced Internet users.

ACKNOWLEDGEMENTS
The author would like to thank to Vojtech Svatek for his comments and help. The research has been partially supported by the FRVS grant no. 501/G1.

References

1. Baker L.:*Search Engine Users Prefer Two Word Phrases*, Search Engine Journal `http://www.searchenginejournal.com/index.php?p=238`
2. Berners-Lee T., Hendler J.,Lassila O.:*The semantic web*. Scientific American, May 2001.
3. Cimiano P. et al.: *Learning Taxonomic Relations from Heterogeneous Evidence*
4. Cimiano, P. and Staab S.:*Learning by googling*. SIGKDD Explor. Newsl. 6, 2 (Dec. 2004), 24-33.
5. Ciravegna F. et al.:*Learning to Harvest Information for the Semantic Web*, Proceedings of the 1st European Semantic Web Symposium, Heraklion, Greece, May 10-12, 2004
6. Etzioni O. et al.:*KnowItNow: Fast, Scalable Information Extraction from the Web*, Proceedings of Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing, p.563–570, October 2005
7. Fellbaum C.: *WordNet, an electronic lexical database*, MIT Press, 1998.

8. Hearst M. A.: *Automatic Acquisition of Hyponyms from Large Text Corpora*. In Proceedings of the Fourteenth International Conference on Computational Linguistics, pages 539–545, Nantes, France, July 1992
9. Kavalec, M, Svatek, V.: *Information Extraction and Ontology Learning Guilded by Web Directory*, Lyon 21.07.2002 26.07.2002. In: AUSSENAC-GILLES, Nathalie, MAEDCHE, Alexander (ed.). Workshop 16. Natural Language Processing and Machine Learning for Ontology Engineering. Lyon : University Claude Bernard, 2002, s. 3942.
10. Kruswitz U.: Intelligent document retrieval : exploiting markup structure, Dordrecht : Springer 2005, ISBN - 1-4020-3767-8
11. Navigli R., Velardi P.:*Structural Semantic Interconnections: A Knowledge-Based Approach to Word Sense Disambiguation*, IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 27, no. 7, pp. 1075-1086, July 2005.
12. Parent S., Mobasher B., and Lytinen S.: *An adaptive agent for web exploration based on concept hierarchies*. In Proceedings of the International Conference on Human Computer Interaction. New Orleans, LA, August 2001
13. Pedersen S., et al.:*Wordnet::similarity - measuring the relatedness of concepts*. In Appears in the Proceedings of the Nineteenth National Conference on Artificial Intelligence (AAAI-04), 2004.
`http://citeseer.ist.psu.edu/644388.html`
14. Porter M.: *Porter Stemmer Algorithm*, [online],
`http://tartarus.org/~martin/PorterStemmer/`
15. Ratnaparkhi A.: Adwait Ratnaparkhi's Research Interests, [online],
`http://www.cis.upenn.edu/~adwait/statnlp.html`.
16. Van Rijsbergen C.J.: Information retrieval (second edition), Chapter 3, Butterworths, London, 1979.
17. Sanderson M., Croft B.:*Deriving concept hierarchies from text*,[online]
`citeseer.ist.psu.edu/cimiano03deriving.html`
18. Weiss S.M. et al.: *Text Mining - Predictive Methods for Analyzing Unstructured Information*. Springer, 2005, ISBN 0-387-95433-3.

GeKon – Applying Novel Approaches to GIS Development

Tomáš Richta

Department of Computer Science, FEE, CTU - Czech Technical University in Prague,
Karlovo nám. 13, 708 33, 121 35 Prague 2, Czech Republic
richtt1@fel.cvut.cz

Abstract. This paper describes a few ideas concerned with geographical information systems (GIS) development. Those ideas come from a GIS development project named GeKon, which is now held on the Department of Computer Science at CTU. The first of them deals with a huge semantic gap between the complex structure of real world and its representation in GIS. Some papers describe this problem as a consequence of wide usage of procedural programing languages and relational database management systems in contemporary GIS development. The object-oriented approach is usually recommended as a better way of constructing such systems. We demonstrate here our findings in object modelling, programing and data management achieved in GeKon project. The other idea shows the fact, that not always the term object-oriented is understood in the same way. People usually have in mind some specific programing language and its structures, instead of the real problem and its solution. In this paper we want to clarify those misleadings and try to describe requirements, that should be fulfilled to achieve expected benefits of this approach. We also introduce next supposed steps in the GeKon project.

Keywords: GIS, semantic gap, object-orientation, data modelling, Smalltalk, OODBMS

1 Introduction

At the beginning, we have to slightly describe geographical information systems (GIS) evolution process to reader be more familiar with questions connected with this area of research. Present GIS were principally constructed as tools for computer based maintenance of geo-referenced data formerly kept in cartographical tools, like maps, atlases, cadastral plans, etc. Evolution of such systems probably started with the idea of cartography digitalization drawn ahead by the vision of map producing acceleration, and also faster access to carographical data, including sorting, searching, and other functions that could lead to better utilization of geo-referenced data.

1.1 Present GIS data management

Present problems of GIS systems have probably their origin in the way of GIS data production. The process of cartography digitalization consists of scanning cartographical tool, and then its vectorization by laying vector paths over the lines in the map. The data produced by this process tend to come under three main geometry types:

- spotted, when some interesting spot had been captured, like a church, a tree, a well, etc.
- linear, when some linear data had been captured, like roads, borders, railways, networks, etc.
- planar, where some area of interest had been captured, like cadastre area, city district, etc.

This separation lead to three main data types used in present GIS: points, lines and polygons. Almost all such systems use as its storage device those three elements. One of the first companies in GIS field named ESRI had invented special file format named shapefile (SHP) for storing these data. SHP becomes something like a standard in GIS data management. Probably for better preservation of cartographical information, data were collected separately in thematic parts. It means, that all churches in observed area were vectorized in one point-based shapefile, all roads in one line-based shapefile, etc. This approach lead to very discrete representation of real world. When the map is to be retrospectively reconstructed in GIS, all thematic files has to be layered over themselves to obtain proper cartographical representation. Layers are not connected among themselves, so ie. the information about roads leading to a well is not present.

Further needs of additional information capturing and storage lead to extension of shape format with so called attribute data component like names, numbers, etc. At the same time as this problem arose, the relational approach to data management achieved great concern. So the GIS files were extended with database file (DBF) including attribute data, connected to their geometry representation in shapefiles. Both those files were stored separately. Relational tables consist of domains, and records, so the real world information has to be separated and simplified into these parts to be stored in DBF. This approach also lead to more discrete GIS data quality and omitting of more complicated information.

Both stated approaches have not changed since the beginning of GIS. The only novel approach is to store SHP and DBF files into relational database management system and reach them through the SQL. But this way only leads to bigger conservation of the problem.

1.2 Semantic gap

Wikipedia describes semantic gap as a mismatch caused by some conflicts emerging in layered systems, when a high level of abstraction need to be translated

to lower, more concrete artifacts [1]. When we try to imagine our modern informatical world as a very complex layered system, we see the real world structure, semantics, and topology on one side and its computer based representation in information systems on the other side. Between them, there are multiple sensors, but not only mechanic ones, but also human senses as a main set of tools used in present digitalization process. Rapant defines the term of geoinformatics as a scientific discipline dealing with quality, behavior, and reciprocal interactions of spatial objects, phenomena, and processes by force of their digital models combined with information technologies [2]. When a GIS may be able to serve as a tool for better understanding of the real world, its way of storing real data must be as natural as possible. But present GIS with their fragmented data model, layered representation, and relational simplicity are not able to provide it. For example, when we want to describe a village, we need to prepare a number of layers covering all fields of our interest, ie. buildings, roads, fields, gardens, forests, sewages, electrical wring, gas pipelines, churches, wells, pubs, shops, bus stations, etc. For each theme, we need a relational table blueprint that describes whether there are points, lines, or polygons, and what attributes we want to store along with geometry. This leads to strenuous process of uniformization of gathered information and quanta of discrete data production. All houses must be described by the same group of attributes, and the information whether house is a pub is stored somewhere else, than the number of its stories. Also when we want to combine together villages belonging to some administrative area, we need to flatten out all the villages thematic layers to be coherent and thus comparable together. This process isn't very natural and its results do not exactly cover described original. And that is why the semantic gap emerges.

2 Object-oriented approach

Merunka recommends an object-oriented approach (OOA) as a good solution of semantic gap in information systems development [3]. He states that OOA solves this problem by introducing the idea of higher level of abstraction in software development with emphasis on modularity, reusability, and standardization. He also summarizes main criteria of object-oriented system:

1. Data and its functionality are encapsulated in one logical entity, which is called object.
2. Objects communicate by sending messages to each other.
3. Objects are able to inherit their properties from other objects.
4. Objects are collected in classes together with other similar objects.
5. Different objects are able to react to the same message, which is called polymorphism.
6. Objects could also have other relationships like composing, dependency, or delegation.
7. Methods are programs consisting of operations over object data. Methods are components of objects.

8. Object identity is independent on object data.

Comparing OOA ideas with GIS problems gives a new horizon to their solution. Our natural way of perception of material aspect of the real world is that everything is an object, and now with OOA everything is possible to be an object in our computer memory. What we only need is to start using OOA in instant. No more thinking about geometrical representation of objects or the way how to store them into tables.

Of course, this is not the novel idea. Mitrovic and Djordjevic-Kajan described in 1996 OOA as natural paradigm for highly complex domains, especially because it maintains a direct correspondence between real world and application objects [4]. Kofler in in 1998 in his PhD. thesis about large 3D GIS databases recommended OOA as an obvious choice for any new GIS [5]. Chance et al., key developers of GE Smallworld GIS, describe OOA as highly effective when applied to the requirements of GIS [6]. The question may arise, why nothing has changed yet, even if those articles had already been written within past ten years.

2.1 Object-oriented modelling

Looking for some indications of OOA emerging in GIS, we could find a few papers describing object modelling in GIS, especially when the 3rd dimension has to be introduced to those systems. For example, Nebiker wrote in 2003 about his fully object-oriented model for 3D geo-objects [7]. Also Kolbe and Groeger work on their unified standard for 3D city models that is strictly based on OOA [8].

2.2 GIS development projects

There are also some projects, that have already implemented parts of GIS using OOA. One of them is already cited in Koflers PhD. thesis [5] where he described a few tests of GIS database implemented over two examples of object-oriented database management systems - ObjectStore and O2. Kofler used as a platform SGI Indigo workstations with MIPS R4400 CPU at 250 MHz and 128 Mbyte main memory, which is not very strong but it is adequate to the year of publishing. He states that OODBMS has bigger performance demands than traditional file systems, but still recommends them as a best solution for GIS database.

Balovnev et al. implemented software for 3D/4D geo-scientific applications developement named GeoToolKit [9]. They used C++ and ObjectStore as core technology tools. GeoToolKit is a class library for the storage and retrieval of spatial objects within an object-oriented database. Developers state that their approach lead to separation of focus on the geoapplication semantics and the need of spatial objects assembly from multiple relational tables. They also mention the reduction of the code written, improvement of its understandability, and they descibe some interesting applications implemented with the use of GeoToolKit library.

One of the other projects is GeoViewer implemented by Lurie et al. in 1997 [10]. Its an object-oriented GIS framework with optimized spatial geometry representation providing transparent linkage to data objects. GeoViewer is written in Smalltalk with small amount coded in C. Minimal documented configuration for running GeoViewer is Sun Sparc 10 with 64MB of RAM or Pentium 166 with 64MB of RAM. This project is remarkable because it incorporates our main ideas about design of GIS, such modelling natural relationships between the geodata, or independent representation of objects and its geometric representation. Taking the year of publication into account, the measure of innovation ideas is stunning. Also the question arises, why this is not the main approach in nowadays GIS.

2.3 GeKon project

Because we see all of those projects very important as a way to improve present GIS applications, we have also opened research project concerning OOA used in GIS development. The name of this project is GeKon an it was started as a semestral work in one of our Software engineering courses. Members of the development team were three undergraduate students - Ivo Kondapaneni, Petr Novosad, Jiri Verunek, and author himself. Ivo Kondapaneni had to leave our project later for his other academic duties, so project extent had to be partially narrowed.

GeKon system is now able to load data from shapefiles and display it on the computer screen. It is also possible to zoom displayed data and move over it. As a test case the city GIS problem was chosen and as a development platform we used Squeak Smalltalk dialect. In Fig. 1. we introduce our object model.

This model shows the GeKon structure - space subdivision, geometry and city object model. In the space subdivision part we formed our idea that it was necessary to maintain some sophisticated indexed structure that allowed fast searching. We use two trees here. First one is responsible for logical subdivision of the city (city districts, basic settlement units, and counting districts) and it is implemented with collections. Second one is responsible for searching the physical space within the counting district and it is implemented as a R-tree. Loaded objects are primarily classified according to the logical space subdivision and then inserted into R-tree of relevant counting district.

Geometry part of our model is responsible only for the geometric representation of objects. Now it contains only necessary representations - point, line and polygon, expanded with 3D solid representations. Each shape has its own bounding box which is stored in space subdivision structures.

The logical object structure of city is the only demonstrative example of part of the city configuration. Its main purpose is to show how we need to model the reality - naturally. We expect that in the future this part of the GIS will be constructed in some visual metamodeller directly by the GIS user at the moment of data import. This approach assumes strict clear disjunction of the data and its representation.

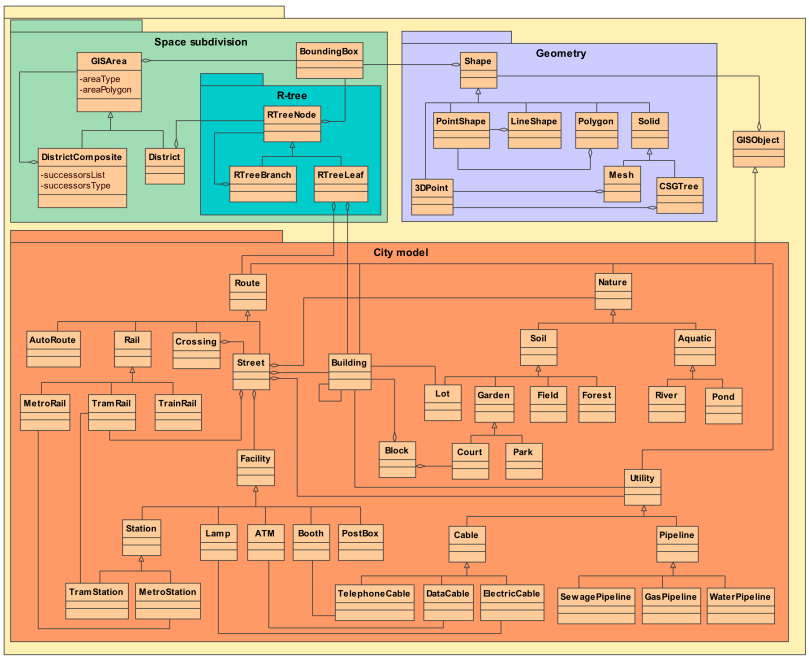

Fig. 1. GeKon object model

Visualization was partly implemented using OpenGL, but after loss of I. Kondapaneni we had to choose easier way. So we used Squeak morphic visualization system, which is sufficient, but lacks possibilities to improve graphics performance in the future like redrawing using textures etc. So we plan to redevelop OpenGL rendering engine for GeKon in future.

Data management has not been covered yet, we are storing all data in Smalltalk image. In future, we plan to incorporate OmniBase as a database management system, which ensures native storage and retrieval of objects. We also plan to cooperate with another project held on our department named CellStore, which deals with heterogeneous data storage and now serves as native XML database.

During the development we used sample data from the ICIP (Institute of City Informatics - Prague), covering the Josefov city district. The average loading time was 1749ms per 1MB and the refreshing time about 320ms, which is not bad. But we see these results orientational only, because the sample is too small to give evidence of the GeKon capabilities. Further we want to use bigger data collections for testing, but they were not available at the moment.

3 Conclusion and future work

In this paper we described semantic gap problem between the real world and its computer based representation. This problem was described from the GIS point of view. GIS are still very tightly coupled with the old fashioned representation of the real world by using files and relational tables. We discussed here OOA as one of the approaches that are able to overcome semantic gap. Some interesting projects concerning object modelling and object-oriented implementations of GIS systems were slightly introduced. In the end of our paper, we also presented our own project GeKon, in which we developed the prototype of object-oriented GIS. Now we want to summarize discovered requirements for the GIS development and also planned work in the GeKon project.

3.1 GIS development requirements

Based on our experience in GeKon system development and also on the knowledge from previously cited papers we strongly recommend following rules to be applied in OOA GIS development:

- Separate geometric representation from object itself to be independent on its shape.
- Use R-trees or other indexing structures for fast searching in space and efficient data retrieval.
- Pay attention to very fast rendering algorithms to prepare the most comfortable environment for users.
- Use metamodeller controlled by the user to obtain the logical structure of the place of interest.
- Use strictly pure non-hybrid object-oriented language to avoid programmers cheating (for details see [3]).

3.2 Further steps in GeKon project

In future, we want to continue in development of GeKon system on our department, partly in the form of dissertation, diploma and other thesis, partly as student semestral projects. Fig. 2. describes the roadmap of planned work.

References

1. Wikipedia, the free encyclopedia, *encyclopedia*, `www.wikipedia.org`.
2. Rapant P.: Zaklady geoinformatiky I. (Geoinformatics fundamentals I.), *course lecture*, Ostrava, The Czech Republic, 2005.
3. Merunka V.: Objektove orientovany pristup k projektovani informacnich systemu (Object oriented approach to information systems development), *habilitation thesis*, Department of Information Engineering, CUA in Prague, The Czech Republic, 2005.

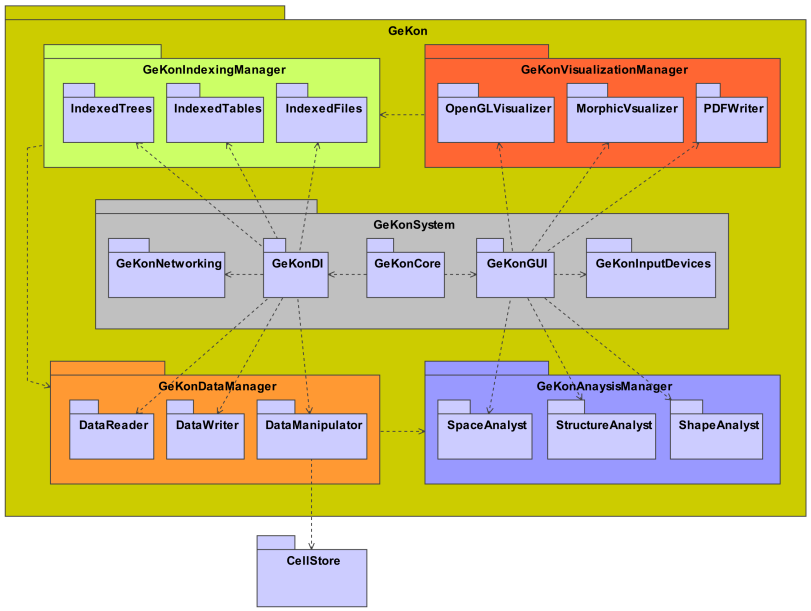

Fig. 2. GeKon structure model

4. Mitrovic A., Djordjevic-Kajan S.: OO paradigm meets GIS: a new era in spatial data management,*invited paper*, presented at YUGIS'96, Belgrade, Yugoslavia, 1996, `http://www.cosc.canterbury.ac.nz/tanja.mitrovic/`
5. Kofler M.: R-trees for Visualizing and Organizing large 3D GIS Databases, *PhD. Thesis*, TU Graz, Austria, 1998.
6. Chance A., Newell R.G., Theriault D.G.: Smalworld GIS: An Object-Oriend GIS - Issues and Solutions, *Smallworld GIS white paper*, 2000, `http://www.logis.ro/downloads/`
7. Nebiker S.: Support for visualization and animation in scalable 3D GIS environment - motivation, concepts and implementation, *scientific paper*, International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences, Vol. XXXIV-5/W10, 2003.
8. Kolbe T.H., Groeger G.: Towards unified 3D city models, *article in:* Proceeding if the ISPRS Comm. IV Joint Workshop on "Challenges in Geospatial Analysis, Integration and Visualization II"', Stuttgart, 2003.
9. Balovnev O., Breunig M., Cremers A.B., Shumilov S.: GeoToolKit: Opening the access to object-oriented geo-data, *scientific paper*, Interoperating Geographic Information Systems, Boston: Kluwer Academic Publishers, 1999.
10. Lurie G.R., Korp P.A., Christiansen J.H.: A Smalltalk-based Extension to Traditional Geographic Information Systems, *students paper*, `http://www.dis.anl.gov/geoviewer/`, 1997

A Comparison of Element-based and Path-based Approaches to Indexing XML Data*

Michal Krátký, Radim Bača

Department of Computer Science, VŠB – Technical University of Ostrava
17. listopadu 15, 708 33 Ostrava–Poruba
{michal.kratky,radim.baca}@vsb.cz
Czech Republic

Abstract. The mark-up language XML (Extensible Mark-up Language) is recently understood as a new approach to data modeling. A well-formed XML document or a set of documents is an XML database and the associated DTD or schema specified in the language XML Schema is its database schema. Implementation of a system enabling us to store and query XML documents efficiently (so called native XML databases) requires a development of new techniques that make it possible to index an XML document in a way that provides an efficient evaluation of a user query. Most of XML query languages are based on the language XPath and use a form of path expressions for composing more general queries. In the paper we compare element-based and path-based approaches to indexing XML data. In the case of element-based approaches query is evaluated step by step. Each step produces a lot of elements which may be refused in the next evaluation step. In the paper we show that the previously published multi-dimensional path-based approach overcomes conventional element-based approaches.

Key words: indexing XML data, XPath, element-based approach, path-based approach, multi-dimensional data structures

1 Introduction

The mark-up language *XML* (*Extensible Mark-up Language*) [20] is recently understood as a new approach to data modelling [16]. A *well-formed* XML document or a set of documents is an XML database and the associated DTD or schema specified in the language *XML Schema* [23] is its database schema. Implementation of a system enabling us to store and query XML documents efficiently (so called *native XML databases*) requires a development of new techniques [16, 5].

An XML document is usually modelled as a graph the nodes of which correspond to XML elements and attributes. The graph is mostly a tree (we consider no attribute `IDREFS` now). To obtain specified data from an XML database a

* Work is partially supported by Grant of GACR No. 201/06/P113.

number of special query languages have been developed, e.g. *XPath* [22], and *XQuery* [21]. A common feature of these languages is a possibility to formulate paths in the XML graph. Such a path is a sequence of element or attribute names from the root element to a leaf. Regular expressions provide a valuable method for paths specifications. In fact, most of XML query languages are based on the XPath language that uses a form of path expressions for composing more general queries. The XPath defines a family of 13 *axes*, i.e. relationship types in that an actual element can be associated to other elements represented in the XML tree. The family of axes defined in the XPath is designed to allow the set of graph traversal operations that are seen to be atomic in XML document trees.

In the past, there were many considerations about use of existing relational or object-relational DBMSs for storing and querying XML data. Since a tree is accessed during evaluation of a query, conventional approaches through the conventional database languages SQL or OQL fail or they are not too efficient. Consequently, a form of indexing is necessary.

Recently there are several approaches to indexing XML or, more general, semistructured data. Some of them are based on a traditional *relational technology* (e.g. *Lore* [15] and *XISS* [14]), the others use special data structures for representation of XML data like *trie* (e.g. *Index Fabric* [7] and *DataGuide* [17]) or multi-dimensional data structures (e.g. *XPath Accelerator* [9]). The latter approach uses R-trees but also B-trees as database indices in environment of a relational DBMS. As it was expected, R-trees outperforms B-trees in this proposal. The work [7] presents an index over the prefix-encoding of the paths in an XML document tree, in which each leaf u_L of the document tree is prefixed by the sequence of element tags that one encounters during a path traversal from the document root to u_L. A more complete summary of various approaching to indexing XML data is e.g. [6].

In the course of the development of XML databases the need for a benchmark framework has become more and more evident: many different ways to store and query XML data have been suggested in the past, e.g. *XMark* [18] and *XML Data Repository* [19].

From the other point of view we distinguish element-based (e.g. *XPath Accelerator* [9] or *XISS* [14]) and path-based (e.g. [13, 11]) approaches to indexing XML data. In the case of an element-based approach a query is evaluated step by step. Each step produces a lot of elements which may be refused in the next evaluation step.

In the paper we compare XPath accelerator (XPA) element-based approach and the multi-dimensional (MDA) path-based approach. We show that the previously published multi-dimensional path-based approach (e.g. [13, 11]) overcomes conventional element-based approaches. In Section 2 we describe XPA as a typical representative of element-based approaches. In Section 3 we describe MDA to indexing XML data. Section 4 reports on results of experiments for selected XPath queries. In conclusion we summarize the paper content and outline possibilities of a future work.

2 XPath Accelerator (XPA)

XPA [9] is an indexing method for efficient evaluation of XPath queries. XPA is an element-based approach to indexing semi-structured data applying multi-dimensional data structures like R-tree or UB-tree but it can be realized in a relational database as well.

2.1 Model of XML documents

Every XML document can be modelled as a tree where one tree node corresponds to exactly one element or attribute of XML document. For support all XPath axes XPA assigns 5-dimensional descriptor $desc(v)$ to every node v

$$desc(v) = \langle pre(v), post(v), par(v), att(v), tag(v) \rangle .$$

The first attribute in the descriptor is *preorder rank* $pre(v)$ of node v. $Pre(v)$ corresponds to a *document order* of node v. *Document order* is defined as an order in which nodes appear in XML serialization of a document. Otherwise said, *document order* corresponds to the order in which nodes come in sequential reading of text representation of XML document. $Pre(v)$ is assigned to node v before any of its children is visited in sequential reading. The second attribute is *postorder rank* $post(v)$ of node v. This number is assigned to node v after all its children are visited. Let v and v' be evaluated nodes of XML tree. Then

- v' is descendant of $v \quad \Leftrightarrow \quad pre(v) < pre(v') \ \wedge \ post(v') < post(v)$,
- v' is following of $v \quad \Leftrightarrow \quad pre(v) > pre(v') \ \wedge \ post(v') < post(v)$.

Similarly, relations *ancestor* and *preceding* can be evaluated between any two nodes. If every node v of the tree has assigned pair $(pre(v), post(v))$ then we are able to determine four major axes *descendant*, *ancestor*, $following$ and *preceding* for the node v with a single query. We call this node a context node. The determination of axe for a context node means finding all nodes which are in the appropriate relation with the context node.

Example 1 (Evaluation of four major axis for context node).

In Figure 1(a) we can see an XML document modelled as a tree. All nodes are evaluated with the preorder and postorder rank. In Figure 1(b) we see *pre/post* plane divided into four regions where every region corresponds to one axis. The division is made for a context node h. Major axes for the context node h contain the following nodes:

- a, f in axis *ancestor* :: $*$,
- k in axis $following$ :: $*$,
- i, j in axis *descendant* :: $*$,
- b, c, d, e, g in axis *preceding* :: $*$.

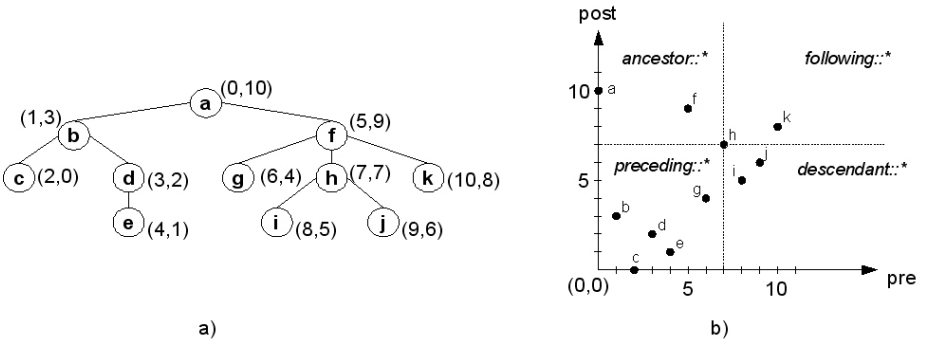

Fig. 1. (a) Evaluation of an XML tree with pairs $(pre(v), post(v))$. (b) Node distribution in $pre/post$ plane and four major axes for a context node h.

Descriptor $desc(v)$ for a node v includes an attribute $par(v)$ for support of axes $parent$, $child$, $following - sibling$ and $preceding - sibling$. The attribute stores the parent's preorder rank of a node v. Boolean attribute $att(v)$ is true if the node v is *attribute*. The attribute att is included to support of *attribute* axis. Finally, attribute tag stores element (or attribute) name id. There is an algorithm which can compute the descriptor $desc$ for every node of an XML tree during single sequential reading of an XML document.

We use a term index for mapping term to its id. Every term which occurs during parsing XML document is faced with term index and mapped to appropriate id. Attribute and element names are stored in XPA index as a part of descriptor $dest$, but also terms in an element content or attribute value should be stored somehow. We decided to use the inverted list [2]. Value of every element or attribute v is parsed into single words which are mapped to ids. We store pairs $(id, pre(v))$ for every word in the inverted list. Consequently, we can retrieve $pre(v)$ of all nodes to be contain searched word.

2.2 Querying in the XPA index

XPA index is done after we map the whole XML document into 5-dimensional space. We resolve *location steps* of XPath query step by step. The *location step* consists of name of the axis, name of the node ($nodeName$) and predicate. The predicate is optional but in the case there is some predicate we have to solve *axis::nodeName* part and predicate separately and then we have to union the results. We designed implementation of XPA so that it is possible to handle nested predicates. Solving *axis::nodeName* part of one *location step* is realized using query upon 5-dimensional space. We find all nodes inside 5-dimensional cube as it is shown in Table 1 in more detail.

Wildcard '*' in Table 1 means that this attribute can have any value to match corresponding axis, but value of attribute tag depends on value of $nodeName$ of

Table 1. Intervals of each attribute for evaluation of corresponding XPath axe.

axe	**pre**	**post**	**par**	**att**	**tag**
child	$(pre(v), \infty)$	$[0, post(v))$	$pre(v)$	false	*
descendant	$(pre(v), \infty)$	$[0, post(v))$	*	false	*
descendant-or-self	$[pre(v), \infty)$	$[0, post(v))$	*	false	*
parent	$[par(v), par(v)]$	$[post(v), \infty)$	*	false	*
ancestor	$[0, pre(v))$	$(post(v), \infty)$	*	false	*
ancestor-or-self	$[0, pre(v)]$	$[post(v), \infty)$	*	false	*
following	$(pre(v), \infty)$	$(post(v), \infty)$	*	false	*
preceding	$[0, pre(v))$	$[0, post(v))$	*	false	*
following-sibling	$(pre(v), \infty)$	$(post(v), \infty)$	$par(v)$	false	*
preceding-sibling	$[0, pre(v))$	$[0, post(v))$	$par(v)$	false	*
attribute	$(pre(v), \infty)$	$[0, post(v))$	$pre(v)$	true	*

location step. When the index applies R-trees or other multi-dimensional data structure retrieving of all nodes inside 5-dimensional cube can be performed by a single range query.

XPath query is evaluated from one context node v_c. XPath query consists of a sequence of *location steps*. Query processing is done in these phases:

1. We obtain a set of nodes S_1 as a result of evaluation of the first *location step* from context node v_c. We set $i = 1$.
2. The set S_i is established as a set of context nodes for the following step.
3. We evaluate $(i + 1)$th *location step* for every context node from the set S_i and the result is a set of nodes S_{i+1}. We increment i by one.
4. Phases 2 and 3 are repeated until the last *location step* of XPath query is evaluated.
5. Set of nodes S_i is the result of the XPath query.

That means running many range queries during every phase 3. With increasing number of *location steps* the execution time of the query increases as well. Size of the set S_i which is created during each *location step* may be much larger then the size of the XPath query result. Such inefficiency leads to unnecessary execution time overhead.

3 Multi-dimensional Approach to Indexing XML Data

In [13, 11, 12] MDA was introduced. This path-based approach applies to indexing XML data paged and balanced multi-dimensional data structures like *UB-trees* [3], *R-trees* [10], *R*-trees* [4], and *BUB-trees* [8].

3.1 Model of XML documents

As mentioned above an XML document may be modelled by a tree, whose nodes correspond to elements and attributes. String values of elements or attributes or empty values occur in leafs. An attribute is modelled as a child of the related element. Consequently, an XML document may be modelled as a set of paths from the root node to all leaf nodes. Note, unique number $id_U(u_i)$ of a node u_i (element or attribute) is obtained by counter increments according to the *document order* [9]. Unique numbers may be obtained using an arbitrary numbering schema. Of course, document order must be preserved.

Let $\mathcal{P}$ be a set of all paths in a XML tree. The path $p \in \mathcal{P}$ in an XML tree is sequence $id_U(u_0), id_U(u_1), \ldots, id_U(u_{\tau_P(p)-1}), s$, where $\tau_P(p)$ is the length of the path p, s is `PCDATA` or `CDATA` string, $id_U(u_i) \in D = \{0, 1, \ldots, 2^{\tau_D} - 1\}$, τ_D is the chosen length of binary representation of a number from domain D. Node u_0 is always the root node of the XML tree. Since each attribute is modelled as a super-leaf node with `CDATA` value, nodes $u_0, u_1, \ldots, u_{\tau_P(p)-2}$ represent elements always.

```
<!DOCTYPE books [
  <!ELEMENT books(book)>
  <!ELEMENT book(title,author)>
  <!ATTLIST book id CDATA #REQUIRED>
  <!ELEMENT title(#PCDATA)>
  <!ELEMENT author(#PCDATA)>
]>
```

```
<?xml version="1.0" ?>
<books>
  <book id="003-04312">
    <title>The Two Towers</title>
    <author>J.R.R. Tolkien</author>
  </book>
  <book id="001-00863">
    <title>The Return of the King</title>
    <author>J.R.R. Tolkien</author>
  </book>
  <book id="045-00012">
    <title>Catch 22</title>
    <author>Joseph Heller</author>
  </book>
</books>
```

Fig. 2. (a) DTD of documents which contain information about books and authors. (b) Well-formed XML document valid w.r.t DTD.

A labelled path lp for a path p is a sequence $s_0, s_1, \ldots, s_{\tau_{LP}(lp)}$ of names of elements or attributes, where $\tau_{LP}(lp)$ is the length of the labelled path lp, and s_i is the name of the element or attribute belonging to the node u_i. Let us denote the set of all labelled paths by $\mathcal{LP}$. A single labelled path belongs to a path, one or more paths belong to a single labelled path. If the element or attribute is empty, then $\tau_P(p) = \tau_{LP}(lp)$, else $\tau_P(p) = \tau_{LP}(lp) + 1$.

Example 2 (Decomposition of XML tree to paths and labelled paths).
In Figure 2 we see an example of an XML document. In Figure 3 we see an XML tree modelling the XML document. We see that this XML document contains paths:

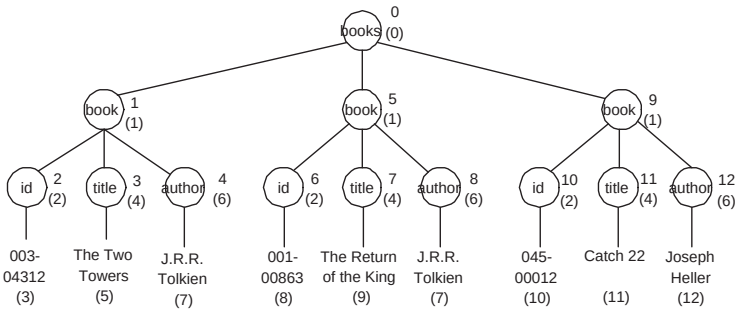

Fig. 3. Example of XML tree with unique numbers $id_U(u_i)$ of elements and attributes u_i and unique numbers $id_T(s_i)$ of names of elements and attributes and their values s_i (values in parenthesis).

– `0,1,2,'003-04312'`; `0,5,6,'001-00863'` ; and `0,9,10,'045-00012'` belong to the labelled path `books,book,id`,

– `0,1,3,'The Two Towers'`; `0,5,7,'The Return of the King'`; and `0,9,11, 'Catch 22'` belong to the labelled path `books,book,title`,

– `0,1,4,'J.R.R. Tolkien'`; `0,5,8,'J.R.R. Tolkien'`; and `0,9,12,'Joseph Heller'` belong to the labelled path `books,book,author`.

Definition 1 (point of n-dimensional space representing a labelled path).

*Let $\Omega_{LP} = D^n$ be an n-dimensional space of labelled paths, $|D| = 2^{\tau_D}$, and $lp \in \mathcal{LP}$ be a labelled path $s_0, s_1, \ldots, s_{\tau_{LP}(lp)}$, where $n = max(\tau_{LP}(lp), lp \in \mathcal{LP}) + 1$. **Point of n-dimensional space representing a labelled path** is defined $t_{lp} = (id_T(s_0), id_T(s_1), \ldots, id_T(s_{\tau_{LP}(lp)})) \in \Omega_{LP}$, where $id_T(s_i)$ is a unique number of term s_i, $id_T(s_i) \in D$. A unique number $id_{LP}(lp_i)$ is assigned to lp_i.* ■

Definition 2 (point of n-dimensional space representing a path).
*Let $\Omega_P = D^n$ be an n-dimensional space of paths, $|D| = 2^{\tau_D}$, $p \in \mathcal{P}$ be a path $id_U(u_0), id_U(u_1), \ldots, id_U(u_{\tau_{LP}(lp)}), s$ and lp a relevant labelled path with the unique number $id_{LP}(lp)$, where $n = max(\tau_P(p), p \in \mathcal{P}) + 2$. **Point of n-dimensional space representing path** is defined $t_p = (id_{LP}(lp), id_U(u_0), \ldots, id_U(u_{\tau_{LP}(lp)}), id_T(s)) \in \Omega_P$.* ■

We define three indexes:

1. **Term index**. This index contains a unique number $id_T(s_i)$ for each term s_i (names and text values of elements and attributes). The unique numbers can be generated by counter increments according to the document order.

We want to get a unique number for a term and a term for a unique number too. This index can be implemented by the B-tree.

In Figure 3 we see the XML tree with unique numbers of terms in parenthesis.

2. **Labelled path index**. Points representing labelled paths together with labelled paths' unique numbers (also generated by counter increments) are stored in the labelled path index.

   In Figure 3 we see that the document contains three unique labelled paths `books,book,id`; `books,book,title`; and `books,book,author`. We create points `(0,1,2)`; `(0,1,4)`; and `(0,1,6)` using id_T of element's and attribute's names. These points are inserted into a multi-dimensional data structure with id_{LP} `0`, `1`, and `2`.

3. **Path index**. Points representing paths are stored in the path index.

   In Figure 3 we see unique numbers of elements. Let us take the path to the value `The Two Towers`. Relevant labelled path `book,book,title` has got id_{LP} 1 (see labelled path index). We get point `(1,0,1,3,5)` after inserting unique numbers of labelled path id_{LP}, unique numbers of elements id_U and term `The Two Towers`. This point is stored in a multi-dimensional data structure.

An XML document is transformed to points of vector spaces and XML queries are implemented using a multi-dimensional data structure queries. The multi-dimensional data structures provide a nature processing of *point* or *range queries* [3]. The point query probes if the vector is or is not present in the data structure. The range query searches all points in a query box $T_1 : T_2$ defined by two points T_1, T_2.

3.2 Queries for values of elements and attributes

Now, implementation of a query for values of elements and attributes and query defined by a simple path based on an ancestor-descendent relation will be described. Query processing is performed in three phases which are connected:

1. **Finding unique numbers id_T of query's term in the term index**.
2. **Finding labelled paths' id_{LP} of query in the labelled path index**. We search the unique numbers in a multi-dimensional data structure using point or range queries.
3. **Finding points in the path index**. We find points representing paths in this index using range queries. Now, we often want to retrieve (using labelled paths and term index) names or values of elements and attributes.

In Figure 4(a) we see that we can model the query (a) as a tree. Consequently, we can create the range queries in the same way as the XML tree is decomposed to vectors of multi-dimensional spaces.

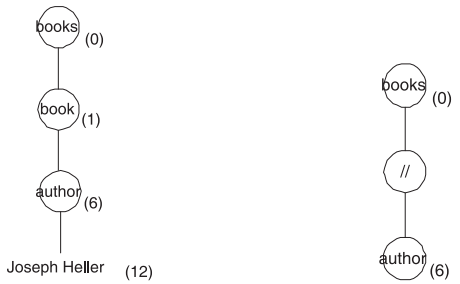

Fig. 4. Trees modelling XPath queries for values of elements or attributes: (a) `/books/books[author='Joseph Heller']` (b) `/books//author`.

Example 3 (Evaluation plan of the XPath query `/books/book[author="Joseph Heller"]`).

By the query we want to retrieve all books written by `Joseph Heller`:

1. Find id_T of terms `books`, `book`, `author`, and `Joseph Heller` in the term index.
2. Find a unique number id_{LP} of the labelled path `books,book,author` in the labelled path index, which was transformed to the point representing the labelled path. We retrieve $id_{LP} = 2$ of labelled path by the point query `(0,1,6)`.
3. Create two points defining a query box, which searches points relevant to this query. The query box is defined by points `(2,0,0,0,12)` and `(2,`max_D`,`max_D`,`max_D`,12)`, where max_D is the maximal value of domain D of space Ω_P. id_{LP} of the labelled path retrieved during the last phase is located in the first points' coordinates. id_T of term `Joseph Heller` is located in the last points' coordinates. Since, we search points with arbitrary values of 2^{nd}–4^{th} coordinates, the first point contains the minimal values of multi-dimensional space's domain and the second point contains the maximal values of the domain.

We need to distinguish labelled paths and paths belonging to element or attribute. We solve it using flags added to points. Similarly, we can solve indexing of more XML documents, which can be valid w.r.t different schema. Hence, the multi-dimensional approach is hopeful for implementation of a native XML database.

4 Experimental results

In our experiments we compare XPA element-based approach with MDA path-based approach. We show the element-based XPA is less effective than MDA.

Both approaches are based on multi-dimensional data structures (R-tree [10] and Signature R-tree [12], respectively). The framework *ATOM* [1] is applied in our implementation of data structures. Although compared approaches are very different we can compare some same parameters (e.g. DAC). Consequently, we show the main disadvantage of element-based approaches. Single steps are evaluated step by step during a query evaluation. Each step produces a lot of elements which may be refused in the next evaluation step. We show the number of the refused elements is rather large in the case of XPA.

In our experiments[1] we use the XMARK collection [18]. The collection contains one file of the size 111MB. It includes 2,082,854 elements. Table 2 shows statistics of XPA indices. In Table 3 tested queries are put forward. These queries were selected wilfully. The first one includes 180 elements in the result, whereas the second one includes 7.5× elements more than the first query, that is 1,350.

Table 2. Statistics of XPA indices (element index, inverted list, and term index)

	XPA index	**Inverted list**	**Term indexes**
Tree level	4	3	1 - 4
Number of items	2,081,550	8,130,422	376,906
Number of inner nodes	1,378	1,574	1,038
Number of list nodes	31,980	71,638	6,559
Average filling [%]	74.8	65.9	61
Size of inner node [B]	2,028	2,044	
Size of leaf node [B]	2,048	2,048	
Item size of inner node [B]	40	16	
Item size of leaf node [B]	24	12	
Dimension	5	2	

Table 3. Two XPath queries evaluated in our experiments

Query	**XPath query**	Result Size
Q1	`/site/closed_auctions/closed_auction/annotation/description/parlist/listitem/parlist/listitem/text/emph/keyword/`	180
Q2	`/site/regions/africa/item[location='United']`	1,350

Tables 5 and 6 show results of evaluation of queries Q1 and Q2, respectively, in XPA. Tables includes surveyed parameters for each *location step*. In Table 4 such parameters are described.

Inefficiency of element-based approach is obvious in the difference between **Nodes** and **Useful** values. In the case of Q1 55,383 elements are retrieved but

[1] The experiments were executed on an Intel Pentium ®4 2.4Ghz, 1GB DDR400, under Windows XP.

Table 4. Surveyed parameters during query evaluation in XPA

Nodes	Number of nodes in the result set after evaluation of one *location step*
Useful	Number of nodes which leads to at least one node in the next *location step*
Time	Time for processing the step
DAC	Number of access in indices

the result contains only 180 elements. In the case of Q2 27,493 elements are retrieved but the result contains only 1,350 elements.

Table 5. Statistics of query Q1 evaluated with XPA

Step	**Nodes**	**Useful**	**Time [s]**	**DAC**
site	1	1	0.02	5
closed_auctions	1	1	0	5
closed_auction	9,750	9,750	1.9	1,386
annotation	9,750	9,750	4.6	50,594
description	9,750	2,934	5	50,252
parlist	2,934	2,934	4.64	49,773
listitem	8,512	1,713	1.9	15,448
parlist	1,713	1,713	4.02	43,114
listitem	4,964	4,964	1.02	8,872
text	4,964	1,890	2.11	24 999
emph	2,864	173	1.97	24,806
keyword	180	**180**	0.95	14,070
Sum	**55,383**	36,003	28.27	283,324

Table 6. Statistics of query Q2 evaluated with XPA

Step	**Nodes**	**Useful**	**Time**	**DAC**
site	1	1	0	5
regions	1	1	0	5
africa	1	1	0	5
item	550	550	0.08	27
location ∼= 'United'	26,940	**1,350**	3.34	5,673
Sum	**27,493**	1,903	3.48	5,716

Table 8 shows results of queries Q1 and Q2 in the case of MDA. Surveyed parameters are put forward in Table 7. We can see the time and DAC of query evaluation is lower than in the case of XPA. The advantage of MDA is obvious.

Table 7. Surveyed parameters during query evaluation in MDA

DAC_t	Number of access in term index
DAC_p	Number of access in labeled path index
DAC_{lp}	Number of access in path index
DAC	Whole number of access in indices
Time	Time of query evaluation

Table 8. Statistics of queries Q1 and Q2 evaluated with MDA

	Q1	**Q2**
DAC_t	211	60
DAC_{lp}	1	144
DAC_p	723	2,400
DAC	934	2,604
Time [s]	0.49	0.26

5 Conclusion

In the paper we compare XPA element-based approach and MDA path-based approach. In the case of an element-based approach a query is evaluated step by step. Each step produces a lot of elements which may be refused in the next evaluation step. Results of our experiments prove the previously published MDA path-based approach overcomes conventional element-based approaches. In our future work, we would like further to improve the abilities and the efficiency of MDA. In particular, we are going to develop an implementation of another complex XML querying such XPath and XQuery query languages defined it. We would like to use data types described by XML Schema for querying and develop an efficient implementation of approximate querying of XML documents.

References

1. Amphora Research Group (ARG). Amphora Tree Object Model (ATOM), `http://arg.vsb.cz/`, 2006.
2. R. Baeza-Yates and B. Ribiero-Neto. *Modern Information Retrieval*. Addison Wesley, New York, 1999.
3. 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*, Lecture Notes in Computer Science. Springer–Verlag, 1997.
4. N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger. The R*-tree: An efficient and robust access method for points and rectangles. In *Proceedings of the 1990 ACM SIGMOD*, pages 322–331. ACM Press, 1990.
5. R. Bourret. XML and Databases, 2001, `http://www.rpbourret.com/xml/XMLAndDatabases.htm`.
6. A. B. Chaudhri, A. Rashid, and R. Zicari. *XML Data Management: Native XML and XML-Enabled Database Systems*. Addison Wesley Professional, 2003.

7. B. Cooper, N. Sample, M. J. Franklin, G. R. Hjaltason, and M. Shadmon. A Fast Index for Semistructured Data. In *Proceedings of the 27th International Conference on Very Large Data Bases (VLDB'01)*, pages 341–350. Morgan Kaufmann, 2001.
8. R. Fenk. The BUB-Tree. In *Proceedings of 28rd VLDB International Conference on Very Large Data Bases (VLDB'02), Hongkong, China*. Morgan Kaufmann, 2002.
9. T. Grust. Accelerating XPath Location Steps. In *Proceedings of the 2002 ACM SIGMOD, Madison, USA*. ACM Press, June 4-6, 2002.
10. A. Guttman. R-Trees: A Dynamic Index Structure for Spatial Searching. In *Proceedings of the 1984 ACM SIGMOD International Conference on Management of Data, Annual Meeting, Boston, USA*, pages 47–57. ACM Press, June 1984.
11. M. Krátký, J. Pokorný, T. Skopal, and V. Snášel. The Geometric Framework for Exact and Similarity Querying XML Data. In *Proceedings of First EurAsian Conference, EurAsia-ICT 2002, Shiraz, Iran*, volume 2510 of *Lecture Notes in Computer Science*. Springer–Verlag, October 27-31, 2002.
12. M. Krátký, J. Pokorný, and V. Snášel. Implementation of XPath Axes in the Multi-dimensional Approach to Indexing XML Data. In *Current Trends in Database Technology, International Workshop on Database Technologies for Handling XML information on the Web, DataX, Int'l Conference on Extending Database Technology (EDBT 2004)*, volume 3268 of *Lecture Notes in Computer Science*. Springer–Verlag, 2004.
13. M. Krátký, J. Pokorný, and V. Snášel. Indexing XML data with UB-trees. In *Proceedings of Advances in Databases and Information Systems, ADBIS 2002, 6th East European Conference, Bratislava, Slovakia*, volume Research Commmunications, pages 155–164, September 8-11, 2002.
14. Q. Li and B. Moon. Indexing and Querying XML Data for Regular Path Expressions. In *Proceedings of 27th International Conference on Very Large Data Bases (VLDB'01)*. Morgan Kaufmann, 2001.
15. J. McHugh, S. Abiteboul, R. Goldman, D. Quass, and J. Widom. Lore: A Database Management System for Semistructured Data. *ACM SIGMOD Record*, 26(3):54–66, 1997.
16. J. Pokorný. *XML: a challenge for databases?*, pages 147–164. Kluwer Academic Publishers, Boston, 2001.
17. J. W. R. Goldman. DataGuides: Enabling Query Formulation and Optimization in Semistructured Databases. In *Proceedings of 23rd International Conference on Very Large Data Bases (VLDB'97)*, pages 436–445. Morgan Kaufmann, 1997.
18. A. R. Schmidt, F. Waas, M. L. Kersten, D. Florescu, I. Manolescu, M. J. Carey, and R. Busse. The XML Benchmark. Technical Report INS-R0103, CWI, Amsterdam, The Netherlands, April, 2001, `http://monetdb.cwi.nl/xml/`.
19. University of Washington's database group. The XML Data Repository, 2002, `http://www.cs.washington.edu/research/xmldatasets/`.
20. W3 Consortium. Extensible Markup Language (XML) 1.0, W3C Recommendation, 10 February 1998, `http://www.w3.org /TR/REC-xml`.
21. W3 Consortium. XQuery 1.0: An XML Query Language, W3C Working Draft, 12 November 2003, `http://www.w3.org/TR/xquery/`.
22. W3 Consortium. XML Path Language (XPath) Version 2.0, W3C Working Draft, 15 November 2002, `http://www.w3.org/TR/xpath20/`.
23. W3 Consortium. XML Schema Part 1: Structure, W3C Recommendation, 2 May 2001, `http://www.w3.org/TR/xmlschema-1/`.

Comparison of Native XML Databases and Experimenting with INEX

Petr Kolář and Pavel Loupal

Dept. of Computer Science and Engineering
FEE, Czech Technical University
Karlovo náměstí 13, 121 35 Praha 2
Czech Republic
kolarp3@fel.cvut.cz, loupalp@fel.cvut.cz

Abstract. The aim of the article is to summarize and compare approaches of design and architecture of native XML databases. We discuss our results accomplished by utilizing the INEX data set in two open source database systems - eXist and Apache Xindice. There is also a basic performance comparison outlined as a basis for discussion about suitability for particular database system and for our consecutive experiments.

1 Introduction

XML documents can be stored in a native XML database. Storage of semi-structured data in a native XML database (NXD) has an advantage in the fact that it has a regular structure but the structure varies enough that what means that mapping this structure into a relational database results in either a large number of columns with null values which wastes spaces or a large number of tables which is inefficient. Another advantage of storing data in a native XML database is the retrieval speed. It is much faster to retrieve data from a native XML database than relational database.

The term *native XML database* is used in different ways by various groups. For our purposes we consider the XML:DB definition – but, to distinguish from XML-enabled databases, we require a native XML database to have the following two properties as well:

- The XML data model (either in the XML Infoset or the XQuery/XPath Data Model) is the fundamental logical data model both used internally by the database and exposed to database users when XML is the data type.
- The XML data model is the fundamental unit of physical storage of all XML data, without mapping to a different data model.

This narrowed definition means that XML is more than an externalized data type - it is how the data is handled both logically and physically. The data is represented as XML right down to its physical storage schema on the disk. This model is the best for efficient searching of the XML data.

2 Comparison of Exist and Xindice XML Native Database

Due to limited space we mention only basic attributes and features of two database systems in following table. In our work we consider Xindice XML database version 1.0 [1] and eXist XML database version 1.0-dev-20060124 [2]. We would like also test Timber or Sedna database, but we decided not to test these databases. Both Timber and Sedna database accept only load of one XML document into database container.

Feature	eXist	Xindice
Technology	Java	Java
Data storage	B+-trees and paged files. Persistent DOM	Natively as indexed text files, Hoffman codes
Binary files	No	No
Transaction Support	No	No
Authorization	Unix like, permissions at collection and document level	No Support
Supported Standards	XPath/XQuery, XUpdate, Xinclude/XPointer	XPath, XUpdate, AutoLinking
APIs	XML:DB	XML:DB, command line
Client GUI	Yes	No
Indices	Structural, Fulltext, Range	

3 Experiments, basic performance comparison

3.1 INEX Dataset

For our experiments we use the INEX XML data set. The INEX data set (we use version 1.4) has 536MB of XML data. It is exactly 12,107 articles from 6 IEEE transactions and 12 journals from years 1995 to 2002. Pictures are not included – data set consists only of XML formated text.

Data set is organized in a file structure. Root directory consists of two subdirectories – *dtd* (holds structure information - DTD specification article element) and *xml*. Each journal/transaction has its own two-letter named subdirectory inside xml directory. Journal/transaction is further divided into the directories by the year of publication. Finally each article is stored in an individual xml file, which name consists of a letter followed by four-digit number and xml suffix.

In average each article contains 1,532 XML nodes, where the average depth of node is 6.9. See [5] for detailed characteristics of data set.

3.2 XPath

XPath [3, 4] is a language for finding information in an XML document – navigating through elements and attributes in an XML document. XPath is a major

element in the W3C's XSLT standard - and XQuery and XPointer are both built on XPath expressions. So an understanding of XPath is fundamental to a lot of advanced XML usage.

We prepared set of XPath queries in following categories:

Selecting nodes. XPath uses path expressions to select nodes in an XML document – e.g. $/article$ or $/article/fm/hdr/hdr1/crt/issn$. Queries 1 to 3 in Table 1.

Predicates. Predicates are used to find a specific node or a node that contains a specific value. Predicates are always embedded in square bracket. E.g. $/article/bdy/sec[1]$ or $/article/bdy/sec[position() < 3]$. Queries 4 to 11 in Table 1.

Selecting Unknown Nodes. XPath wildcards can be used to select unknown XML elements – e.g. $/*/*[@*]$. Queries 12 to 14 in Table 1.

Selecting Several Paths. By using the | operator in an XPath expression we can select several paths – e.g. $//article/fm/hdr|//article/bdy/sec$. See queries 15 and 16 in Table 1.

4 Results

We measured duration time of each query five times. Then we discarded the largest and the smallest value and counted arithmetic mean.

The time needed to load INEX data set into database was 25 minutes for Xindice and 97 minutes for eXist. The data on filesystem took 600 MB for Xindice and 1300 MB for eXist. Our hardware configuration was based on a personal computer with Intel Celeron 1.7 Ghz processor, 512MB RAM and Windows XP(SP2) operating system. INEX XML data set in version 2003 (1.4). Detailed information about the data set and its structure is shown in Section 3.1.

4.1 Summary

Our results do not meet our expectations – Xindice has totally failed in our experiments. With regard to our results this database system is impracticable for more extensive XML data sets. Althought we tried to create indices for all elements and attributes but without any significant improvement.

Most of XPath queries running over Xindice returned an empty result set – it seems that Xindice does not fully support the XPath 1.0 specification but only its limited subset. On the contrary, eXist showed much better behavior. This can be induced by its automatically generated structural index that is very efficient. eXist has also an user friendly GUI for both database management and ad-hoc query processing.

No.	Query	Records retrieved	Query duration time [s] eXist	Xindice
1	$/article$	12104	1,3	230
2	$/article/fm/hdr/hdr1/crt/issn$	11666	2,2	98
3	$//issn$	11666	1,3	447
4	$/article/bdy/sec[1]$	11955	1,9	NA
5	$/article/bdy/sec[last()]$	11955	5,6	NA
6	$/article/bdy/sec[last()-1]$	11019	5,8	NA
7	$/article/bdy/sec[position()<3]$	22974	8,1	NA
8	$//sec[@type]$	868	1,0	more than 10 min
9	$//sec/p/ref[@type='bib']$	108496	81,3	NA
10	$/article/fm/hdr/hdr2/pdt[yr='1995']$	1623	2,6	NA
11	$/article/fm/hdr/hdr2/pdt[yr='1995' and mo='Spring']$	72	4,0	NA
12	$/article/*$	58472	164,3	NA
13	$/*/*[@*]$	49	352,0	NA
14	$//fig[@*]$	52857	70,6	NA
15	$//article/fm/hdr \mid //article/bdy/sec$	77487	8,6	NA
16	$//article/fm/hdr/hdr1 \mid //article/fm/hdr/hdr2$	24208	3,8	NA

Fig. 1. Results of given queries

5 Conclusion

The aim of our experiment – to test some of native XML databases and perform basic performance comparison – was in principle not successful. We were not able to import the INEX data set into all proposed native XML databases. Therefore we carried out only basic tests for the eXist and Xindice databases. Our results show that for further experiments we should consider only the eXist database. Xindice can be used just as an example of a basic native XML database.

We would like to perform further comparisons among other native XML databases. Also, we plan to add some of non-native (or hybrid) XML databases.

References

1. Apache Xindice - Native XML database. http://xml.apache.org/xindice.
2. eXist Native XML database. http://exist.sourceforge.net/.
3. D. Chamberlin, A. Berglund, and e. a. Scott Boag. XML Path Language (XPath) 2.0, September 2005. http://www.w3.org/TR/xpath20/.
4. J. Clark and S. DeRose. XML Path Language (XPath) 1.0, November 1999. http://www.w3.org/TR/xpath.
5. Fuhr, N., Gvert, N., Kazai, G., Lalmas, M. Initiative for the evaluation of xml retrieval (INEX), 2003.

Author Index

Editors: V. Snášel, K. Richta, J. Pokorný

Publisher: Department of Computer Science
FEECS, VŠB-TU Ostrava,
17. listopadu 15,
708 33 Ostrava-Poruba,
Czech Republic

Title: DATESO 2006

Place, year, edition: Ostrava, 2006, 1st

Page count: 132

Print: TiskServis Jiří Pustina
Gen. Sochora 1764,
708 00 Ostrava-Poruba,
Czech Republic

Impression: 200

Not for sale

ISBN 80-248-1025-5