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

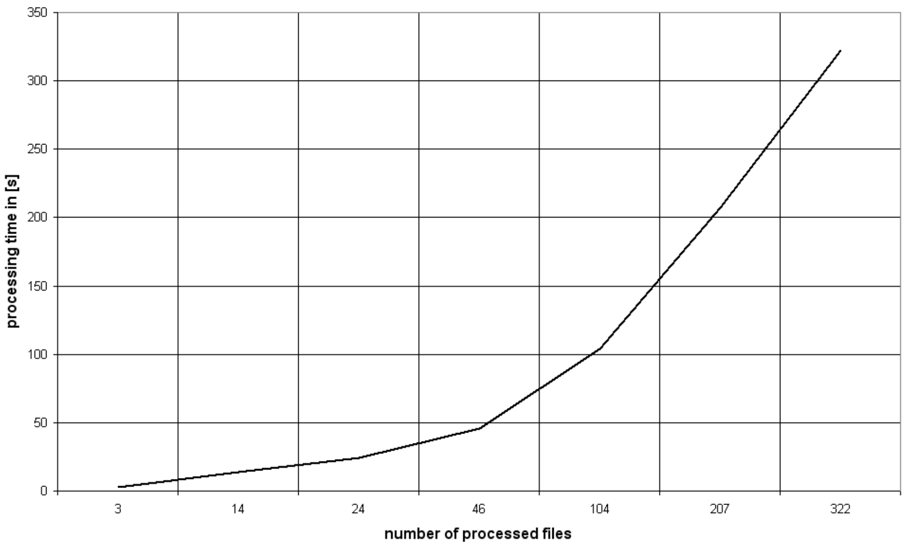

Fig. 1. XMLStoreExt — time dependency on amount of processed files

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

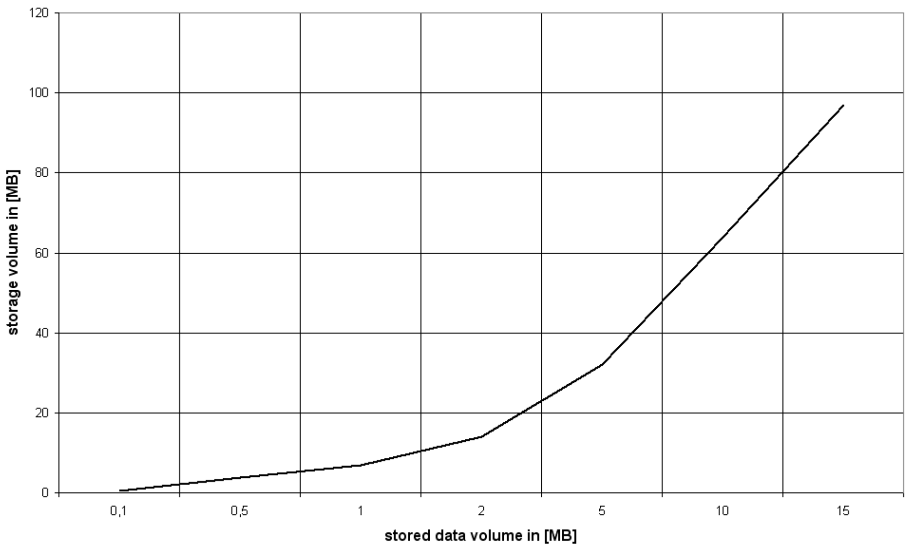

Fig. 2. XMLStoreExt — storage size dependency on size of stored XML data

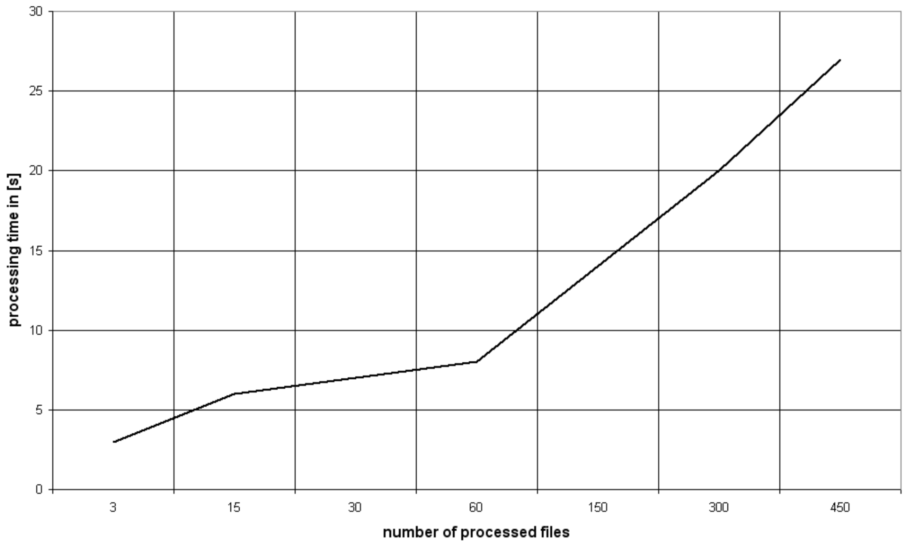

Fig. 3. JAXBStore — time dependency on amount of processed files

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

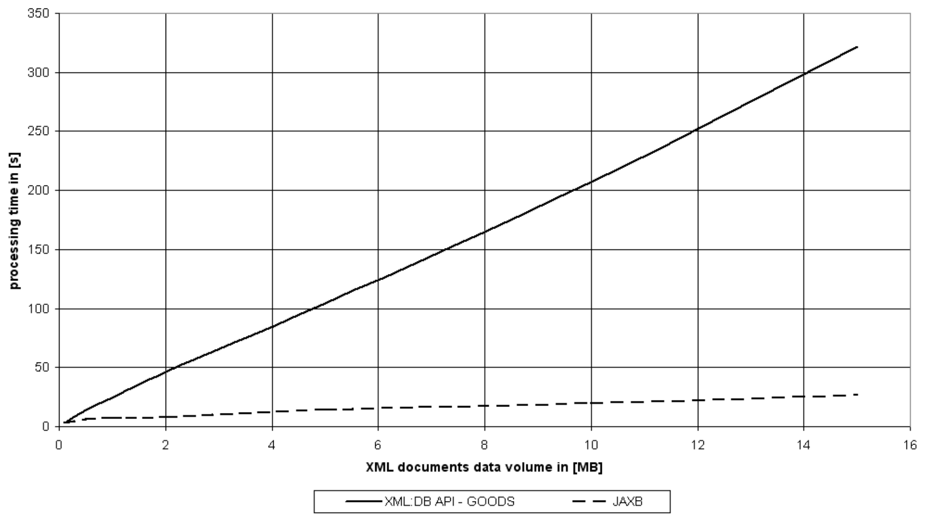

Fig. 4. Efficiency measurement of both referred technologies

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