# EFFICIENT STORAGE OF XML – A COMPARATIVE STUDY

By

UMA-CHINMAYEE NIRMAL

Bachelor of Science

University College of Engineering

Osmania University

Hyderabad, India

2001

Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
MASTER OF SCIENCE
May, 2005

EFFICIENT STORAGE OF XML – A COMPARATIVE

STUDY

Thesis Approved:

Dr. Blayne E. Mayfield

Thesis Adviser

Dr. Johnson P. Thomas

Dr. G. E. Hedrick

Dr. A. Gordon Emslie

Dean of the Graduate College

TABLE OF CONTENTS

CHAPTER PAGE

I. INTRODUCTION ..... 1

1.1 Problem Statement ..... 2
1.2 Thesis Outline ..... 2

II. LITERATURE REVIEW ..... 3

2.1 XML ..... 3
2.1.1 Features of XML ..... 4
2.1.2 Is XML a database? ..... 5
2.2 Data, Documents and Databases ..... 7
2.2.1 Role of Databases ..... 8
2.2.2 Role of Java ..... 8
2.3 Storing and Retrieving Data ..... 9
2.3.1 Integral Storage Systems ..... 9
2.3.2 Full-text search Systems ..... 10
2.3.3 Relational Databases ..... 10
2.3.4 Native XML Databases ..... 14
2.4 Storing and Retrieving Documents ..... 15
2.4.1 File Systems ..... 15
2.4.2 Relational DBMS ..... 15
2.4.3 Native XML Databases ..... 16
2.4.4 Object based data/document ..... 17
2.4.5 Using Java Objects ..... 19

III. DESIGN AND IMPLEMENTATION ISSUES ..... 20

3.1 Implementation Platform and Environment ..... 20
3.2 Scope of Study ..... 20
3.3 Input Parameters ..... 21
3.3.1 XML Input ..... 21
3.3.2 XML Schema ..... 21
3.3.3 Schema – Table Mapping ..... 22
3.4 Design of the Simulation ..... 22
3.4.1 File Systems ..... 22
3.4.2 Relational Databases ..... 23

3.4.3 Native XML Databases ....29
3.4.4 Java Objects ....32

IV. EVALUATION OF THE SIMULATION....35

4.1 Comparative Study....35
4.1.1 Bulk Loading ....36
4.1.2 Path Traversals ....37
4.1.3 Casting ....39
4.1.4 Missing Elements ....39
4.1.5 Ordered Access ....40
4.1.6 Joins ....41
4.1.7 Containment and full-text search ....42
4.2 Performance ....42
4.2.1 Execution Time ....43
4.2.2 Document Size Complexity ....44
4.3 Performance of Queries ....46
4.3.1 Insert Operations ....46
4.3.2 Select Operations ....47
4.3.3 Update Operations ....48
4.3.4 Delete Operations....49
4.4 Comparison of Features ....50
4.4.1 Storage and retrieval in RDBMS ....50
4.4.2 Storage and retrieval in NXD ....50
4.4.3 Storage and retrieval using JAXB....51
4.5 Discussion ....52
4.6 Conclusion ....53
4.6.1 Strengths ....53
4.6.2 Weak Points ....54

V. SUMMARY AND FUTURE WORK....55

5.1 Summary ....55
5.2 Future Work ....56

REFERENCES ....50

APPENDICIES ....54
Appendix A - Abbreviations ....55
Appendix B - Trademark Information ....56
Appendix C - Results ....57
Appendix D - Code Listing ....58

LIST OF TABLES

TABLE PAGE

I Feature-wise performance comparison of results .....36

II Results of query processing comparison .....66

LIST OF FIGURES


| FIGURE | | PAGE |
|---|---|---|
| 1 | Data Exchange using XML | 4 |
| 2 | XML Document | 5 |
| 3 | Example of XML Query | 13 |
| 4 | Example of Stored Procedure using OPENXML | 25 |
| 5 | Example of Insert using Updategram | 26 |
| 6 | Retrieval using FOR XML - Example | 28 |
| 7 | Example of XML Template | 29 |
| 8 | XML document – example | 31 |
| 9 | Java Architecture for XML Binding - overview | 33 |
| 10 | JAXB unmarshal process | 33 |
| 11 | JAXB marshalling into XML | 34 |
| 12 | Performance Graph I – Query Execution Time Complexity | 43 |
| 13 | Performance Graph II – Document Size complexity – single document | 44 |
| 14 | Performance Graph III – Document Size complexity – collection | 45 |
| 15 | Performance Graph IV – Execution Time – Insert Statements | 46 |
| 16 | Performance Graph V – Execution Time – Select Statements | 47 |
| 17 | Performance Graph VI – Execution Time – Update Statements | 48 |
| 18 | Performance Graph VII – Execution Time – Delete Statements | 49 |

# CHAPTER I

## INTRODUCTION

XML is becoming the data format of choice for a wide variety of information systems solutions. Common applications using XML include document transmission in Business to Business (B2B) systems, message format construction for integration of Internet applications with legacy systems, binding of XML data to visual and non-visual controls, data storage and retrieval and various data manipulation activities within applications. The benefits most often associated with the utilization of XML include platform independence, low cost of entry and the ability to seamlessly share data. XML also can accommodate a variety of data including text, images and sound. However, the excitement over XML does not come without challenges stemming from emerging and conflicting standards, new skill requirements and the management issues caused by the growing amounts of XML data to be managed [10].

A majority of both traditional business applications and Internet-based applications depend on databases. Several different database models exist; however, the majority of them are relational [11]. To maintain data in a database, data must be retrieved and stored in a consistent, reliable and efficient manner. Using existing database infrastructure to manage XML data demands seems logical. However, new XML database products are built to handle XML data demands natively without the baggage of converting to other database structures such as the relational structure. In addition, a

variety of storage strategies, conversion processes, and levels of support for leading database products exist [3].

## 1.1 Problem Statement

The use of XML has been growing at a tremendous rate in the recent years. Also several technologies supporting the storage and usage of XML have evolved providing their proprietary functionalities. A need therefore arises for a systematic study to distinguish these technologies from one another, compare their features and analyze their best usage contexts to promote an easy learning curve to neophyte developers. This study focuses on an unbiased comparison of the current and evolving technologies that support XML storage and processing.

## 1.2 Thesis Outline

This study is a survey and comparative analysis of data storage using databases to store and retrieve XML, using Java objects representing XML. It gives a high-level overview of how to use XML with databases or Java Objects. The study also describes how the differences between data-centric and document-centric XML affect their usage, when used with databases and objects and how XML is used with relational and object oriented databases, Java Objects and the role of native XML databases (stand alone XML databases).

# CHAPTER II

# LITERATURE REVIEW

Chapter II is a brief review of the existing technologies, their usage, their merits and demerits and their influence on the proposed orientation of research.

## 2.1 XML

Extensible Markup Language (XML) is a meta-language (a language for describing other languages) that enables designers to create their own customized tags to provide functionality not available with Hypertext Markup Language (HTML) [31]. XML is not a replacement for HTML. HTML is about displaying information, while XML is about describing information. XML was created to structure, store and exchange information. Unlike HTML, XML cannot be interpreted by an ordinary web browser. A good description of XML is that it is a cross-platform, software and hardware independent tool for transmitting information [32].

XML is a subset of Standard Generalised Markup Language (SGML). That is, every XML document is a conforming SGML document.

### 2.1.1 Features of XML

XML can be used to exchange data. As XML is stored in plain text format, which provides a software- and hardware-independent way of sharing data, converting data to XML can reduce greatly the complexity caused by incompatible data formats by creating data that can be read by many different types of applications.

Figure 1 represents a context where XML is used to exchange data over the Internet.

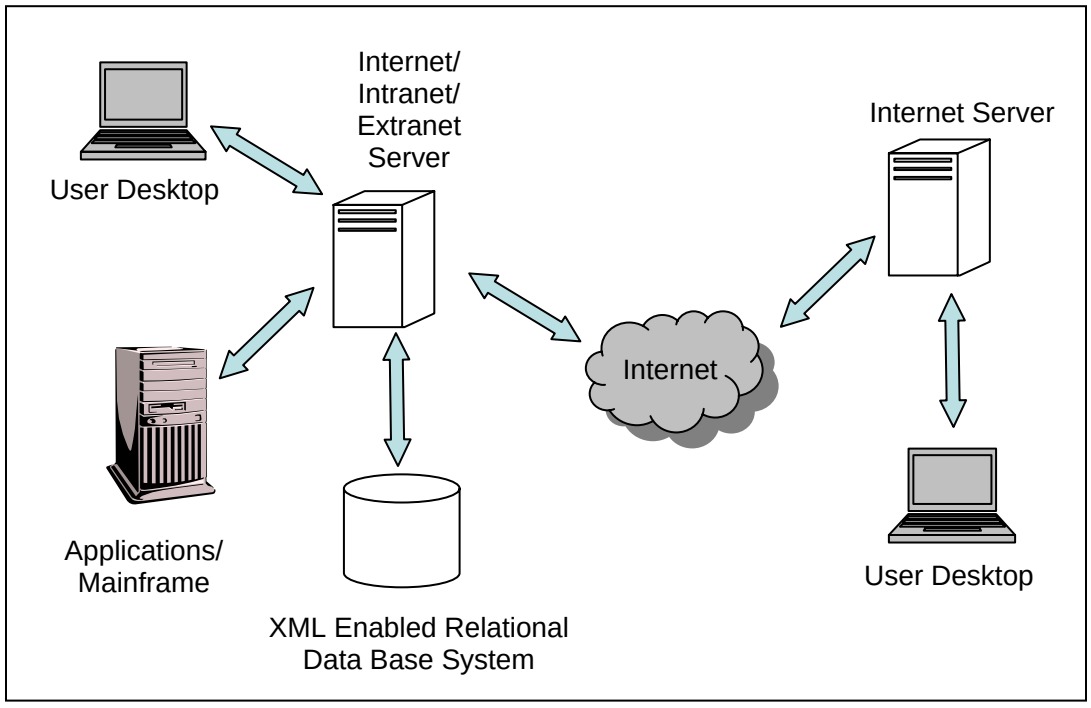


Figure 1. Data Exchange Using XML [12]

XML can also be used to store data in files or in databases. Applications can be written to store and retrieve information from the stored XML and generic applications can be used to display the data. An example of an XML document is shown in Figure 2.

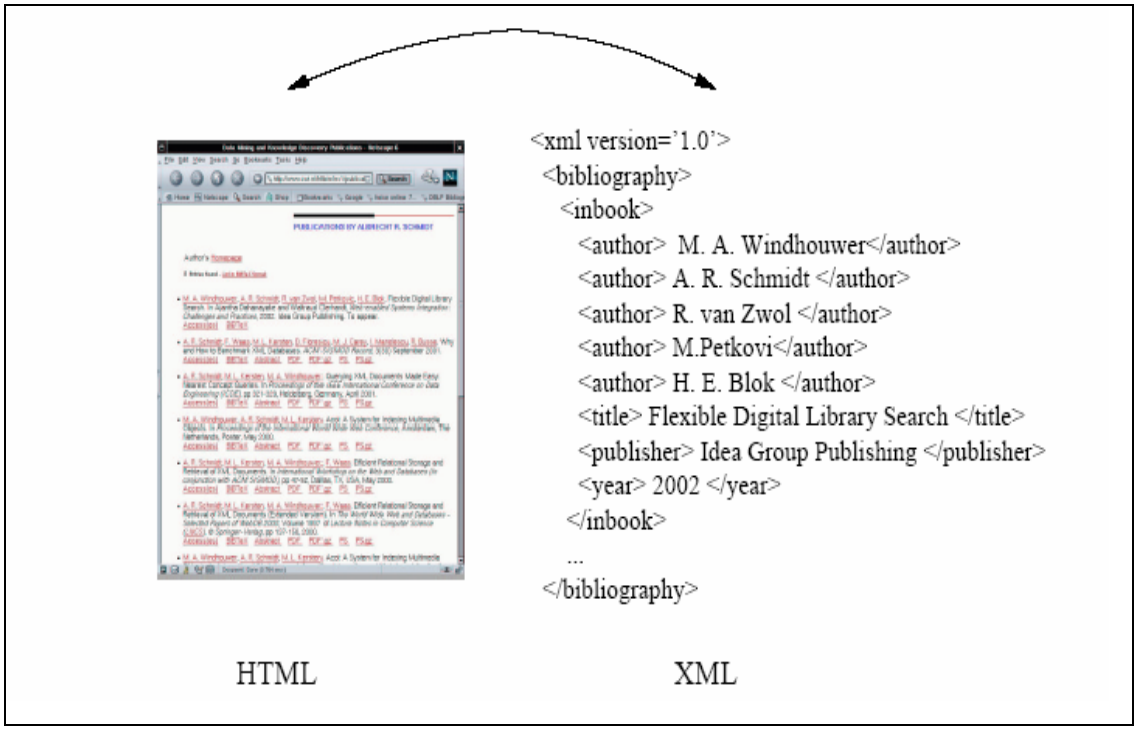


Figure 2. XML document (abstracting from visual representation). [31]

XML can be used to create new languages, for example Wireless Markup Language that is extensively used in handhelds and wireless applications [32].

### 2.1.2 Is XML a database?

An XML document is a database only in the strictest sense of the term: a *database* is a collection of information managed as an asset for the benefit of a community of users [18]. In many ways, this makes an XML file no different from any other file as all files contain data of some sort. As a "database" format, XML has some advantages. For example, it is self-describing (the markup describes the structure and type names of the data, although not the semantics), it is portable (it represents data as Unicode), and it can

describe data as a tree. It also has some disadvantages. For example, it is verbose and access to the data is slow due to parsing and text conversion, as it needs to be formatted for display or otherwise interpreted.

In order for databases to be managed for a community of users, they must exist in a database management system (DBMS) that provides services like caching, space management, locking, backup and restore, transactions and logging. All databases regardless of their data model persuasion benefit from these services. A more useful question to ask would be whether XML and its affiliated technologies constitute a DBMS. On the positive side, XML provides many of the things found in Data Base Management Systems: storage, schemas, query languages, programming interfaces and so on. On the negative side, it lacks many of the things found in “real” DBMSes: efficient storage, indices, security, transactions and data integrity, multi-user access, triggers, queries across multiple documents and so on.

Thus, while it may be possible to use an XML document or documents as a database in environments with small amounts of data, few users and modest performance requirements, this will fail in most production environments, which have many users, strict data integrity requirements, and the need for good performance [17].

XML allows defining nested entries, something that is hard to do in flat files. The real advantage of XML when compared to storing in a traditional database is that the data is portable and this is becoming less of an advantage than it appears, due to the widespread availability of tools for serializing databases as XML, which makes data exchange less formidable.

## 2.2 Data, Documents and Databases

XML products and applications are sometimes characterized as component-centric or data-centric, in contrast to document-centric. Several possible meanings could be intended:

- Distributed storage in contrast to integral storage
- Component-level access as opposed to whole-document access, or
- Both of the above

*Data-centric* XML files are those that use XML as a data transport and are characterized by fairly regular structure, fine-grained data (that is, the smallest independent unit of data is at the level of a character data-only element or an attribute), and little or no mixed content (i.e., no complex element types) [3].

*Document-centric* XML files (usually) are XML documents designed for human consumption and are characterized by little or no regularity in structure, coarser-grained data (that is, the smallest independent unit of data might be at the level of an element with mixed content or the entire document itself), and lots of mixed content [3].

Sean McGrath proposed a more quantitive approach on the xml-dev mailing list. He defined document-oriented XML as "XML in which corpora conforming to schema X exhibit power law distributions of the element types in X" and data-oriented XML as "XML in which corpora conforming to schema X exhibit uniform distributions of the element types in X." [26]

### 2.2.1 Role of Databases

Data-centric information usually is stored in a traditional database which may be a relational, an object-oriented, or a hierarchical database. Data-centric XML information can also be stored into a database by providing third-party middleware or by providing additional capabilities to the DBMS, thus making it XML-enabled.

Document-centric XML, on the other hand, is stored in a database designed especially for storing XML. For example, in an XML (only) database or in an application designed to manage documents and is built on top of a native XML database, which may be a content management system.

### 2.2.2 Role of Java

The rules of data-centric XML and document-centric XML are not absolute. XML data -- especially semi-structured data, can be stored in native XML databases and XML documents can be stored in traditional databases when few XML-specific features are needed. Furthermore, the boundaries between traditional databases and native XML databases are beginning to blur as traditional databases add XML capabilities and XML databases support the storage of document fragments in external databases.

The role of Java in storage and retrieval of both data-centric and document-centric XML is notable. As data is more often than not, managed across platforms, networks and databases, the interface at which its transfer is managed, plays an active role in determining its consistency, security and manageability. As such, Java is a good tool to use, with its platform independent API [5]. When using XML and Java technologies

together, there is greater interoperability formed with other applications both inside and outside of the Enterprise [5].

The remainder of this study discusses the strategies and issues for the storage and retrieval of data, documents, and their interoperability, using various technologies.

## 2.3 Storing and Retrieving Data

Storing and retrieving data is a key measure of the performance of any storage system. As such, storage and retrieval of XML data from a text-based XML file is inefficient as it requires a lot of overhead on the part of the application that uses XML to handle the store, retrieve or search operations. Storage of XML documents using existing data storage tools is discussed in this section.

### 2.3.1 Integral storage systems

Documents have been around far longer than databases, so there are clearly other ways to store and retrieve them than storing in traditional databases.

One way is integral storage, which normally keeps the unparsed document intact and stores it as a Binary Large Object (BLOB) in a database or as a file in the operating system file store. Integral files are typically supplemented with an index database that allows searches based on metadata, such as descriptors, creation and modification dates, authorship, access rights, etc. But if the file store is unmanaged, users can access the files directly and change them. As a result, it can be difficult to maintain the integrity of the index and its consistency with the document files.

There are several products that overcome this problem. They support integral storage by providing managed access and specialized query facilities, either in OS file store or in dedicated repositories

### 2.3.2 Full-text search systems

Another approach to XML document storage and retrieval is the full-text search system. These products focus on searching for documents based on their textual content. Some of them are even XML aware and can take advantage of structural knowledge gained from Document Type Definition (DTD) [2] or schema.

### 2.3.3 Relational databases

Relational databases have been around for so long that the need for a Relational Data Base Management System (RDBMS) to receive and send XML is obvious; that is the essence of using XML as an interchange representation. The most important factor in choosing a database is whether one is using the database to store data or documents. To transfer data between XML documents and a relational database, it is necessary to map the XML document schema (such as DTD, XML Schemas etc.,) to the database schema. Data transfer software is then employed. The software may use an XML query language (such as XQuery, XPath, or proprietary language) or simply transfer data according to the mapping (some equivalent of SQL queries of a database) in which case the structure of the document must exactly match the structure expected by the mapping.

When this is not the case, before transferring data to the database, the document is first transformed to the structure expected by the mapping. Similarly, after transferring data from the database, the resulting document is transformed to the structure needed by the application.

Section 2.3.3.1 gives a brief overview of mapping XML schemas to database schemas.

2.3.3.1 The mapping process Mappings between document schemas and database schemas are performed on element types, and text. They almost always omit physical structure such as entities, Character DATA (CDATA) sections, encoding information, some logical structure such as processing instructions, comments, and the order in which elements appear in their parent. This is more reasonable than it may sound, as the database and application are concerned only with the data in the XML document. For example, in a sales order, it does not matter if the customer number is stored in a CDATA section, an external entity, nor does it matter if the customer number is stored before or after the order date.

One consequence of this is that storing the data from an XML document in a database and then reconstructing the document from that data, also called "round-tripping", often results in a different document. Accepting this modified document for use depends on the application that uses this document and the mapping software being used.

2.3.3.2 Mapping types Two types of mappings are commonly used to map an XML document schema to the database schema: *table-based mappings* and *object-relational mapping* [3].

*Table-based* mappings that are used by many middleware products, transfer data between an XML document and a relational database. They model XML documents as a single table or a set of tables. The table-based mapping is useful for serializing relational data, such as when transferring data between two relational databases. Its obvious drawback is that it cannot be used for any XML documents that do not match the format of the table.

*Object-relational* mappings model data in an XML document as a tree of objects. In this model, XML element types, or mixed content are generally modeled as classes. The model is then mapped to relational databases using traditional object-relational mapping techniques.

2.3.3.3 Query Languages Many products transfer data directly according to the specific data structure or model on which they are built. Because the structure of the XML document often is different from the structure of the database, these products often include or are used with XSLT (eXtensible Stylesheet Language Transformations) [4]. This allows users to transform documents to the structure dictated by the product, before transferring data to the database, as well as the reverse.

2.3.3.3.1 Template Based Query Languages The most common query languages that return XML from relational databases are *Template-based query languages*. In these languages, there is no predefined mapping between the document and the database. Instead, SELECT statements are embedded in a template and the results are processed by the data transfer software. Template-based query languages are used almost exclusively to transfer data from relational databases to XML documents. Although some products that use template-based query languages can transfer data from XML documents to relational databases, they do not use their full template language for this purpose. Instead, they use a table-based mapping [3].

2.3.3.3.2 Structured Query Language based query languages *Structured Query Language (SQL) - based query languages* use modified SQL statements, for example SELECT statements, the results of which are transformed to XML. The simplest of these uses nested statements that are transformed directly to nested XML according to the object-relational mapping. For example, the query in Figure 3 constructs a result set with two columns.

```
SELECT Orders.SONumber,
        XMLELEMENT(NAME "Order",
        XMLATTRIBUTES(Orders.SONumber AS SONumber),
         XMLELEMENT(NAME "Date", Orders.Date),
         XMLELEMENT(NAME "Customer", Orders.Customer)) AS xmldocument
FROM Orders
```

Figure 3. Example of Query [4]

2.3.3.3.3 XML Query Languages Unlike template-based query languages and SQL-based query languages, which can only be used with relational databases,

*XML query languages* can be used over any XML document. To use these with relational databases, the data in the database must be viewed as XML, thereby allowing queries over virtual XML documents. With XML query languages, either a table-based mapping or an object-relational mapping can be used. If a table-based mapping is used, each table is treated as a separate document and joins between tables (documents) are specified in the query itself, as in SQL. If an object-relational mapping is used, hierarchies of tables are treated as a single document and joins, if any, are specified in the mapping. [3]

### 2.3.4 Storing data in Native XML databases

It also is possible to store XML data in a native XML database. *Native XML databases* are databases designed especially to store XML documents. Like other databases, they support features like transactions, security, multi-user access, programmatic APIs, query languages, and so on.

There are several reasons to use a native XML database. The first of these is when data is semi-structured. That is, it has a regular structure, but that structure varies enough that mapping it to a relational database results either in a large number of columns with null values resulting in space wastage or a large number of tables resulting in inefficiency. Although semi-structured data can be stored in object-oriented and hierarchical databases, it also can be stored in a native XML database in the form of an XML document. [6]

## 2.4 Storing and Retrieving Documents

There are two basic strategies to storing a collection of XML documents: store them in the file system or as a Binary Large OBject (BLOB) in a relational database and accept limited XML functionality, or store them in a native XML database.

### 2.4.1 Storing and Retrieving File Systems

If set of documents is simple, such as a small documentation set, the easiest way to store them is in the file system. Tools such as ‘grep’ (to query) and ‘sed’ (to modify) may be used on the document. If simple transaction control is desired, documents can be placed in a version control system.

### 2.4.2 Storing in Relational DBMS

A relational DBMS must offer more than the generic services such as integral storage and full-text searching if it is to support XML. An XML DBMS has requirements that a typical relational DBMS does not. These stem primarily from differences in:

- data models
- applications
- update policies
- schema management

A slightly more sophisticated option is to store documents as BLOBs in a relational database. This provides a number of the advantages found with databases: transactional control, security, multi-user access, and so on. In addition, many relational

databases have tools for searching text and can do such things as full-text searches, proximity searches, synonym searches, and fuzzy searches [11].

### 2.4.3 Storing in Native XML databases

If more features are desired than are offered by one of the systems mentioned above, a native XML database may be considered. The only difference from other systems is that their internal model is based on XML and not something else, such as the relational model.

Native XML databases are mostly useful for storing document-centric XML. This is because native XML databases preserve things such as document order, processing instructions, comments, and entity usage, while XML-enabled traditional databases do not.

Native XML databases also are useful for storing documents whose "natural format" is XML, regardless of what those documents contain. The native XML database offers XML-specific capabilities, such as XML query languages, and usually will be faster at retrieving whole messages.

In short, a native XML database can accept, store, and understand any XML document without prior configuration.

2.4.3.1 XML Database Management Systems A content management system or XML DBMS typically maintains XML documents in dispersed storage; that is, the documents are parsed and their data content, attribute values, and other components are stored in separate records and fields. They are designed for

managing human-written documents such as user manuals and white papers. The database generally is hidden from the user behind a front end that provides features such as:

- Version and access control;
- Search engines;
- XML/SGML editors;
- Separation of content and style;
- Extensibility through scripting or programming; and
- Integration of database data.

The term "*content management system*", as opposed to "*document management system*", reflects the fact that such systems generally allow breaking documents into discrete content fragments, rather than having to manage each document as a whole. Not only does this simplify such things as coordinating the work of multiple writers working on the same document, but also allows assembling entirely new documents from existing components.

### 2.4.4 Storing XML as Object Based Data/Document

In object-oriented approaches (or OO databases), XML documents reside on the database in an object format. For example, each XML element becomes an object. Each attribute in an XML element becomes an attribute in the object. Relationships between elements - that is, their position in the hierarchy, the element sequence, etc. – are retained.

If designed properly, an object-based system retains all document content, including white space. Object based solutions provide good processing characteristics in both text-related and non-text related situations because they directly access the elements, attributes and data.

2.4.4.1 Storing Data/Documents An object-based solution has the ability to treat XML as either data or document. Object-based systems have data management capabilities that are similar to relational systems. Object-based solutions directly models business objects that "represent tangible entities within an application that the users create access and manipulate while performing a transaction" [18]. Business objects within a system are typically persistent and long-lived. They contain business data and model the business behavior. [18] The database easily supports uniqueness and integrity using object identifiers (Object-IDs). For example, changing the street address in a database record is as simple as updating the street address object that defines the street address fragment.

2.4.4.2 Retrieval An object-based system meets the content retrieval requirements. Using the object-based approach, there is the potential of three search methods:

- Use SQL to search the database for specific documents or fragments;
- Use XPATH to search the contents within a specific XML document; or
- Use XQUERY to search XML documents and content.

Since each element is a separate entity, it is fast and efficient to find all occurrences of a specific element within a document or across documents.

### 2.4.5 Object based Solution – Using Java Objects

An object-based solution supports database transactions. Using an object-based approach, a Document Object Model (DOM) implementation can map directly into the database. A DOM is a tree of objects with interfaces for traversing the tree and writing an XML version of it [39]. In fact, when reading an XML file, a DOM tree can be created in memory as specific nodes that can be accessed by the application. This combined with the capability of processing a particular fragment or node of the tree, greatly improves performance, because most of the data stays on the database management system. This solution simplifies resource contention, because fewer records need to be locked. With an object-based approach, it is possible to rely upon the support of a Java Data Base Connectivity (JDBC) API, because JDBC supports object data access as well as relational data access. This means that Java custom mapping can be available to map from the database to Java objects automatically.

Having discussed about the existing technologies and their features, as above, this study explores the strengths and weaknesses of these methods and provides an explanation for those behaviors.

A general program has been devised to test these technologies against one another, the description of which along with results produced are discussed in the subsequent chapters.

# CHAPTER III

# DESIGN AND IMPLEMENTATION ISSUES

## 3.1 Implementation Platform and Environment

The simulation was implemented on the OSU Computer Science Department's Dell Dimension 8200, which is a cutting-edge technology-class computer that represents the common development environment in the industry. The system has 1.0 Giga Byte of RAM. It also has 54.5 Giga Byte of hard disk space. It runs on the Microsoft Windows XP Professional operating system. The development platform included Sun Microsystems Java Runtime Environment JRE 1.4.2_06, Java Software Development Kit J2SDK 1.4.2_06, Apache Software Foundation's Tomcat 5.5 Web Application Server, Apache Software Foundation's Xindice 1.1 B4 Native XML Database and Microsoft SQL Server 2000 SP3 for Windows XP Professional along with SQLXML 3.0 SP3 for Relational database – XML support.

## 3.2 Scope of Study

A careful study and comparative analysis of the existing technologies is pursued. The ideal context for usage of each of the technologies is analyzed, which describes the ideal data, its characteristics, environment, nature of applications and the performance under such or other conditions, for data usage.

This study is limited to establishing the relationship between the modeling and mapping of a given set of data to its most efficient storage mechanism, thereby obtaining its storage and retrieval speed, consistency and efficiency as a measure of performance. This study is also limited by the functionalities provided by the current release versions of the platform and environments used.

## 3.3 Input Parameters

### 3.3.1 XML Input

The data input to this program is in XML format. The data has been obtained from various sources using a common repository [38]. The data is mostly text based but used in various applications such as image processing, web based auctions, conference proceedings documents and financial applications used by banks.

### 3.3.2 XML Schema

The data obtained from the repository has been validated using a W3C XML Schema that was generated for the XML data where DTD was available. The schema generation from a given DTD was carried out using a conversion tool called Trang [7].

### 3.3.3 Schema – Table Mapping

In order to use the XML data with a relational database, MS SQL Server 2000 SP3 in this project, a third party software [1] was used to map the W3C XML Schema with tables in the database.

## 3.4 Design of the Simulation

The simulation was implemented in Java 1.4.2_06, on a Dell Dimension 8200 machine running Windows XP professional operating system.

Various data sets representing extensively used and problematic data were utilized as test data for the research. The data contained content based data (mostly text), some non-textual data, nested data (hierarchical) and hexadecimal

A software program has been developed to utilize the available raw XML data to measure the performance of storage and retrieval of the XML document using a relational database, native XML database and object persistence using Java objects.

### 3.4.1 File Systems

Since storage and retrieval of XML using file systems requires processing on the part of the application using the XML, file systems have not been included in this study. If the processing is done in an environment that has support for parsing and processing XML search, store and retrieve operations, though trivial, can be expensive.

### 3.4.2 Relational Databases

Relational databases and XML documents are both ways to represent data, but they do so differently. For example, in an RDBMS, the metadata is kept in proprietary form whereas in XML it is interspersed with the data in the form of tags, and may also be included in a DTD or a schema definition. Also, RDBMS data is stored in tables and XML data is organized in hierarchies. Besides, the primary languages for accessing data in these two representations – SQL and XPath – reflect these differences. Hence, a mapping from XML to RDBMS is required if the XML document is to be decomposed into the relational format, often called “shredding”.

In this study, a third party tool, Altova GmbH’s Map force [1] was used for the mapping from the XML document schema to RDBMS schema.

The RDBMS used for this study was Microsoft SQL Server 2000 SP3. SQL Server 2000 provides Transact – SQL (T-SQL) language extensions to operate bi-directionally with relational and XML sources. It also provides two system stored procedures, sp_XML_preparedocument and sp_XML_removedocument, that assist the XML to Relational transformation. SQL 2000 Server’s XML access capabilities are further enhanced with SQLXML 3.0 SP3 API, which allows XML formatting both on the client and the server. Prior to SQLXML 3.0, XML formatting was only on the server.

3.4.2.1 Storage Storage of an XML document involves two basic strategies mentioned earlier: integral storage with markup intact and distributed storage across a set of tables. Integral storage preserves the unparsed text, while distributed storage normally parses the document and discards the markup. It is

also possible to store some elements as text while storing others as relational data. MS SQL Server 2000 SP3 provides some such utilities.

The process of pulling an XML document apart and inserting the data into tables is often referred to as shredding. OPENXML is a Transact - SQL (T-SQL) keyword that provides a result set, which can be equated to a table or a view, over an XML document that is retained in memory [28]. OPENXML allows access to XML data as if it were a relational recordset by providing a tabular relational view of the internal representation of an XML document.

Before using OPENXML, SQL Server needs to load the XML document into memory. SQL Server provides the sp_xml_preparedocument stored procedure for doing this. It loads the supplied XML text into an internal DOM (using MSXML2.dll) and returns a handle that can be supplied later to the OPENXML statement.

The handle returned by sp_xml_preparedocument is valid for the duration of the SQL Server connection or until it is removed from memory by calling the stored procedure sp_xml_removedocument. SQL Server stores the loaded documents in its internal cache, but only allows MSXML to use one-eighth of the total memory available. Hence, it is important to call sp_xml_removedocument stored procedure when working with XML documents is finished working with the XML documents. The stored procedure in Figure 4 illustrates this process.

```
DECLARE @xmlDoc text
DECLARE @iDoc int
SET @xmlDoc = '<data> …… </data>'
 -- load XML document into memory
     EXEC sp_xml_preparedocument @iDoc OUTPUT, @xmlDoc
 -- process XML documents with OPENXML here
     OPENXML( @iDoc, RowPattern, [Flags], [WITH (SchemaDeclaration | TableName ) ] )
-- remove XML document from memory
     EXEC sp_xml_removedocument @iDoc
```

Figure 4. Example stored procedure that uses OPENXML statement

When using OPENXML, the nodes in the XML document that map to rows in the new rowset must be identified. This is specified using an XPath expression that identifies an XML node-set.

As MS SQL Server 2000 SP3 does not provide a utility to load XML documents directly using their file paths into queries, a stored procedure was used to extend the capabilities of the existing sp_xml_preparedocument stored procedure to enable reading from a filename.

MS SQL Server 2000 in association with MS Internet Information Services Web Application Server 5.1 also provides a useful functionality of using XML Data Reduced Schemas (XDR) to query SQL data. XDR provides the ability to query and retrieve data over HTTP. Storage using INSERT and UPDATE SQL queries can be made using XML Templates. XML templates are discussed in more detail in Section 3.4.2.2.2.

Updategrams are XML Templates with special tags. Instead of writing SQL Statements to do INSERT, UPDATE, SELECT and DELETE operations, updategrams provide the special tags with functionality that are used like before and after images of the database.

The three main pieces of Updategram are <sync>, <before> and <after>. The <before> and <after> blocks come in pairs to create a single transaction.

These blocks are contained within a <sync> block that defines the transaction. The <before> block is a before image prior to the transaction taking place. The <after> block is the after image, post transaction having taken place. Stated otherwise, the <before> block is the state of the existing data, the <after> block is the state of the data after the transaction occurs. Figure 5 shows the sample code for the INSERT updategram.

```
<ROOT xmlns:updg=”urn:schemas=Microsoft-com:xml-updategram”>
  <updg:sync>
        <updg:before>
        </updg:before>
        <updg:after>
            <Users FirstName=”B” LastName=”Pt” />
        </updg:after>
  </updg:sync>
<ROOT>
```

Figure 5: Example Insert into MS SQL Server 2000 table using insert updategram

Updategrams have not been used in this study as there is no support to read XML files containing the template queries in a single batch process in Java, which was the platform used to conduct the study. However, Microsoft SQLXML 3.0 SP3 API provides support for Microsoft Visual Basic and Microsoft C# Active Data Object (ADO) connections to load the XML processing query files and the corresponding schema files through the SQLXMLBULKLOAD object, that is tied to the Microsoft Win32 Virtual Machine, for generic schema and XML template document processing.

3.4.2.2 Retrieval The retrieval of XML from MS SQL Server 2000 SP3 can be done using one of the two methods in Section 3.4.2.2.1 or 3.4.2.2.2.

3.4.2.2.1 Built-in xml generator functions MS SQL Server 2000 provides the ability to return the results of a query in XML format. This is accomplished by adding the FOR XML clause at the end of the SELECT statement. It is not difficult to use, and the syntax is:

FOR XML *mode* [, XMLDATA] [, ELEMENTS] [, BINARY BASE 64]

- *mode* – this is the only required argument. It specifies how the XML will be returned in the result set. There are 3 values that can be used:
  1. *AUTO* – returns query results as nested XML elements.
  2. *EXPLICIT* – you specify the format of the results.
  3. *RAW* – each row is returned as an XML element
- *XMLDATA* – when specified, the schema is returned along with the results.
- *ELEMENTS* – this argument can only be used in conjunction when mode is *AUTO*. The query results are returned as XML sub-elements.
- *BINARY BASE64* – if the resultset being returned will contain binary data then this attribute specifies what form the results will look like. The binary data is returned as *BASE 64*.

```
SELECT team.Sponsor, rider.RiderName FROM Team, Rider WHERE team.TeamID =
rider.TeamID  FOR XML AUTO ELEMENTS
Results look like:
<Team>
        <Sponsor>Yamaha</Sponsor>
<Rider>
        <RiderName>Damon Bradshaw</RiderName>
</Rider>
<Rider>
        <RiderName>Jeremy McGrath</RiderName>
</Rider>
<Rider>
        <RiderName>David Vuillemin</RiderName>
</Rider>
</Team>
<Team>
<Sponsor>Honda</Sponsor>
<Rider>
        <RiderName>Ricky Carmichael</RiderName>
</Rider>
<Rider>
        <RiderName>Sebastien Tortelli</RiderName>
</Rider>
</Team>
```

Figure 6. Retrieval using FOR XML clause

3.4.2.2.2 Virtual directory and Schema based storage A virtual directory can be created using MS Internet Information Services Web Application Server 5.1 in association with MS SQL Server 2000. XML Data Reduced Schemas (XDR) can then be used to query SQL data. XDR provides the ability to query and retrieve data over HTTP using XML Templates. A template is a valid XML document that can contain a root element, header, parameters, and query statements. Every element within a template is optional, except for the "sql" namespace, which must be defined. A template can have one or more Transact-SQL queries and retrieve the data through a URL, all without exposing any of the underlying database structure. Adding a query to a template is by simply wrapping the query statement within an <sql:query> </sql:query> tag. An example template is provided in Figure 7.

```
<ROOT xmlns:sql="urn:schemas=microsoft-com:xml-sql">
    <sql:query>
            SELECT UserID, FirstName, LastName
            FROM Users
            ORDER BY LastName, FirstName
            FOR XML AUTO
  </sql:query>
</ROOT>
```

Figure 7. Example of an XML template

The template query can be executed by simply typing the "*http://localhost/Users/usertemplate/users.xml"* URL into a web browser on a system where IIS server is installed and the virtual directory is Users.

3.4.2.3 Indexing and Searching Database designers spend considerable effort on indexing, in order to optimize the performance of searches. XML, because of its different data model and mixed content, presents unique challenges in this area. Searching in SQL Server can be done by querying the database using SELECT statements in the regular fashion or using OPENXML statements from an XML document.

### 3.4.3 Native XML Databases

Native XML databases (NXD) or XML DBM Systems are specialized for storing XML data and store all components of the XML model intact. They have an XML document as their fundamental unit of (logical) storage, just as a relational database has a row in a table as its fundamental unit of (logical) storage. A NXD may not actually be a standalone database at all.

3.4.3.1 Storage Native XML DBMSes store XML documents as a unit and create a model that is closely aligned with XML or one of XML's related technologies like the Infoset or DOM. This model includes arbitrary levels of nesting and complexity, as well as complete support for mixed content and semi-structured data. The model is automatically mapped by the NXD into the underlying storage mechanism. The mapping used will insure that the XML specific model of the data is maintained. Once the data is stored, NXDs must be continued in use if a useful representation of data is expected. The database abstracts away all the details of how XML is stored and leaves the developer free to build applications using XML technologies.

In Apache Xindice 1.1 B4, a unique resource identifier for the document is created every time a document is added to a collection and no resource id is specified.

A document ‘fx102.xml’ will be added to the collection ‘/db/data/products’ and will be stored under the key or resource id ‘fx102’, using the command ‘*xindice add_document -c /db/data/products -f fx102.xml’*

3.4.3.2 Retrieval XPath is the current NXD query language of choice. In order to function as a database query language, XPath is extended slightly to allow queries across collections of documents. Unfortunately, XPath was not really designed as a database query language and comes up short in several ways when it is used as one.

To improve the performance of queries, NXDs support the creation of indices on the data stored in collections. These indices can be used to improve the speed of query execution dramatically. In Apache Xindice 1.1b4, the document identified by the key or resource id ‘fx102’ will be retrieved from the ‘/db/data/products’ collection and stored in the file ‘fx102.xml’ using the command “*xindice retrieve_document -c /db/data/products -n fx102 -f fx102.xml”*

3.4.3.3 Indexing and Searching The Xindice indexing system allows defining indices to speed performance on commonly used XPath queries. If no indices are defined queries can still be executed but performance will suffer because the query engine will need to scan the entire collection to create the result node-set every time a query is made. Adding an index is done using “*xindice add_indexer -c /db/data/catalog -n idindex -p product_id*”. Figure 8 shows the XML document containing the product_id element. Adding an index for the product_id element improves product searches based on product_id by making it fast and efficient.

```
<?xml version=”1.0” ?>
   <product>
            <product_id > 120320 </product_id>
            <description> Glazed Ham </description>
   </product>
```

Figure 8. XML document example

### 3.4.4 Java Objects

A Java application that uses XML technologies needs to parse the data objects of XML. The parsing can be done using various parsers that are in vogue, such as Simple API for XML (SAX) or Document Object Model (DOM). A newer parse and bind capable functional API released by Java called as Java Architecture for XML Binding (JAXB) combines the best of both worlds. SAX is an event driven model, whereas JAXB is based on binding of Java objects with XML elements. Data storage capability is higher in JAXB compared to SAX. JAXB applications tend to be faster than DOM as JAXB is specific to a single schema where as DOM can be managed with multiple schemas.

The fundamental concept is based on the fact that JAXB simplifies access to an XML document from a Java program. It allows accessing and processing XML data without having to know XML or XML processing. The JAXB compiler generates a set of Java classes based on the valid schema of the XML file passed to it and creates a content tree of Java objects in memory. The compiler comes with a parser that can be used to validate the XML file based on its schema. The creation of the content tree from the XML file is known as unmarshalling and output of XML from a content tree is known as marshalling. Figure 9 gives a detailed level overview of JAXB.

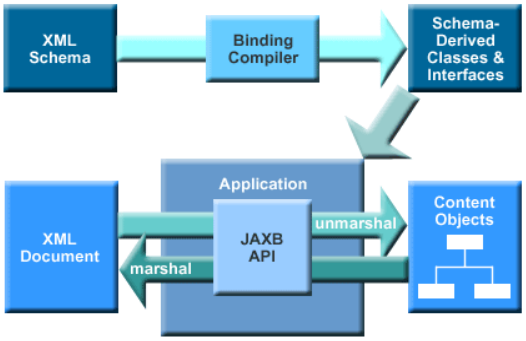


Figure 9: Java Architecture for XML Binding overview [39]

3.4.4.1 Storage Storing XML in Java objects is done by breaking the XML data into chunks or unmarshalling it so that it can be identified by the Java program using it. Unmarshalling an XML document means creating a tree of content objects that represents the content and organization of the document.

The content objects are instances of the classes produced by the binding compiler. In addition to providing a binding compiler, a JAXB implementation must provide runtime APIs for JAXB-related operations such as unmarshalling, marshalling or validation. The APIs are provided as part of a binding framework. The binding framework comprises three packages. The primary package, javax.xml.bind, contains classes and interfaces for performing operations such as unmarshalling, marshalling, and validation. Figure 10 outlines the unmarshalling process of JAXB.

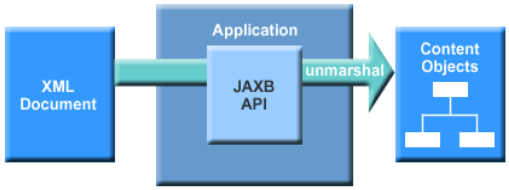


Figure 10: JAXB unmarshal process [39]

3.4.4.2 Retrieval To retrieve XML from the Java content tree objects, it needs to be marshaled. Marshalling is the process of building XML from Java content tree objects. Marshalling is the opposite of unmarshalling. It creates an XML document from a content tree. To marshal a content tree, the appropriate JAXB classes are invoked that handle the output of the objects into XML using a Java output stream handler. Figure 11 outlines the marshalling process of JAXB.

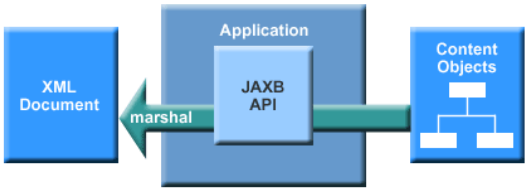


Figure 11: JAXB marshalling into XML [39]

3.4.4.3 Indexing and searching Searching for XML in the JAXB context does not use XPath. Unlike the other two storage mechanisms studied so far, the application using the JAXB classes is expected to handle the search functionality, as the search will not look in the XML document, but in the generated JAXB class collection. As the content tree is loaded in memory, the search is usually faster than with other applications as the application does not have to depend upon database connections and driver communication overheads are eliminated.

# CHAPTER IV

# EVALUATION OF THE SIMULATION

This chapter evaluates the simulation and discusses its performance in terms of storage and retrieval times and efficiency, based on the observations discussed in Section 4.3. The behavior of results produced by various inputs is also discussed.

## 4.1 Comparative Study

A general performance assessment cannot scrutinize differences on a fine level of granularity [35]. Rather, some general challenges were identified, which guided a comprehensive and conclusive performance analysis covering all performance critical aspects of processing of XML.

The comparison is summarized in Table 1. Each of these properties is explained in the section 4.1.1 through to section 4.1.7.

| Property | Relational Databases | Native XML Databases | Java Objects |
|---|---|---|---|
| Preserve Document Order | No (document is shredded) | Yes | No ( dispersed objects ) |
| Support for casting | Yes ( database defined ) | Yes (only standard conversions ) | Yes ( standard and custom data type conversions ) |
| Path Traversal | XPath for queries (Database Implementaion Dependent ) | XPath | Application handled |
| Handles Missing Elements | Yes ( using NULL ) | Yes ( empty elements ) | Yes ( JAXB defined standard data type conversions ) |
| Support for querying over the HTTP | Yes (using virtual directory support of IIS) | Yes (using Servlets on a web application server ) | Yes ( application dependent implementation ) |
| Scalability or Bulk loading | Yes ( platform based ) | Yes ( collection based ) | Yes |
| Full text searches | Yes ( only if database provides support & BLOB/CLOB representation of XML) | Yes | No (document order not preserved ) |
| Construction of large results | Yes | Yes | Yes |
| Multiple Document Search results | Yes (based on database joins) | Yes (collection based) | Yes ( application dependent ) |
| Well Formed XML Parsing Supported | Yes | Yes | Yes |
| Valid XML Parsing Supported | Yes | No | Yes |

Table 1: Feature-wise performance comparison of the technologies used

### 4.1.1 Bulk loading.

The importance of bulk loading data has been repeatedly emphasized when assessing database performance in general. In XML databases, bulk loading currently assumes an important role as no insert/update operations are available and most systems support insert only on a document level. Due to the fact that different structures (models) imply different levels of granularity, while shredding the document, this operation may entail severe costs to setup and maintain indices or constraints.

Bulk loading of XML was conducted using all the three storage technologies. MS SQL Server proved to be the most suitable in terms of sheer bulk. It offered storage of bulk loads of XML using an SQLXML 3.0 SP3 sqlxmlbulkloader object utility in Microsoft Visual Basic and Microsoft C# environments. Extension of sp_xml_preparedocument stored procedure to read from a file currently supports a maximum xml document size of 14 Mega Byte.

Apache Xindice 1.1 b4 NXD does not support a single XML document larger than 3 MB. It was designed for collections of small to medium sized documents as opposed to a single monstrous document. A large document can be split into several smaller documents and grouped together as a collection in order to store in the Apache Xindice 1.1b4 NXD.

Parsing of XML into Java objects using JAXB poses little problem for the size of the document unlike the other two technologies. The JAXB application supporting classes are generated from the JAXB framework by validating against an XML Schema and the processing (unmarshalling, marshalling or validating) is done on the XML document based on the classes generated. This process does not set a limit on the XML document size. As the content tree of objects is created in main memory, the workable size of the document is dependent on the system configuration and available resources.

### 4.1.2 Path traversals

Specifying paths arguably is one of the most basic and natural operations on structured documents. Not only are path expressions useful as a stand alone operation, but also are furthermore ubiquitous building blocks of most complex operations. Efficient

path traversals often cause a trade-off between redundancy, data volume and degree of fragmentation.

XPath's ability to address an XML document at any level of granularity has interesting implications for a DBMS. Nodesets (sets of nodes examined) in XPath are by definition unordered. Since XPath doesn't specify any particular order in which nodes are required to be returned, languages that build on XPath often define additional semantics of their own to take care of ordering. For example, in the case of XSLT, nodesets are always processed in document order.

MS SQL Server 2000 SP3 provides XPath based queries in its OPENXML syntax. The patterns used to identify the nodes to be processed as rows are expressed in XPath syntax. The set of nodes selected by an XPath expression are the nodes remaining after processing each step in order. Usage of XML templates also will result in the processing of nodesets in the document order.

Apache Xindice 1.1 b4 supports XPath and XUpdate. XUpdate makes extensive use of the expression language defined by XPath for selecting elements for updating and for conditional processing. It is designed with references to the definition of XSLT [17]. Hence, the path traversal is in the document order or based on the index order if one is set explicitly by adding to the NXD.

Processing XML using Java objects does not specify any particular node path, unlike relational or XML databases. As the XML is parsed into a tree of objects the path followed is purely based on the access mechanism (logic) of the processing application.

### 4.1.3 Casting

XML is essentially text and as such, queries frequently demand casting to other elementary data types like integers, floats or even user-defined types. String operations are notoriously expensive.

If the XML document is shredded to store into the MS SQL Server 2000 SP3 database, the mapping tool that maps the XML Schema to the RDBMS schema is responsible for the consistent casting of types across the two schemas. All operations performed thereafter convert the data types based on the schema mapping, as and when necessary.

Apache Xindice provides support for casting implicitly as it supports evaluations in XPath search strings that require string to integer and floating point conversions. Other than that, there is no necessary use of casting in Xindice as the data is stored, processed and retrieved as XML.

The JAXB framework on the other hand provides support for Data type conversions both implicitly and explicitly. Implicit conversions take place during the generation of the core content tree object classes by the framework. Explicit conversions are supported by providing external binding files that contain binding customizations, which may also include data type conversions.

### 4.1.4 Missing elements.

The semi-structured nature of XML in general brings about a highly heterogeneous structure of records that under some mappings results in many NULL

values. Apart from strategies to store NULL values in a compact way, we also need efficient methods to query for NULLs as they often represent spots of interest.

Interestingly MS SQL Server 2000 SP3, Apache Xindice 1.1-b4, and JAXB provide support for missing elements, each in their own way. MS SQL Server 2000 SP3 and JAXB need a valid XML document validated by a W3C XML Schema of the XML document in order to process. Hence, missing elements or empty elements are taken into account by the table-mapping and class generation frameworks respectively.

Apache Xindice, unlike the other two does not decompose (shred) the XML document into a non-XML form and hence does not provide an explicit NULL mapping for an empty or missing element.

### 4.1.5 Ordered access.

Order is an omni-present feature when querying XML and affects all aspects of data management. Maintaining order and preserving the implementations may become a severe bottleneck. A sophisticated and flexible treatment of the document order should ensure that it is well integrated into the optimization process up to the degree that it can be ignored when it is not needed, if, specified by the user.

Order management of the documents is important to maintain and preserve a collection of documents. As both MS SQL Server 2000 and Java Architecture for XML Binding decompose an XML input document into their respective proprietary implementations, the document order is not preserved intrinsically. Maintaining the document order can be possible in a relational DBMS when XML is stored as a Character

Large Object (CLOB) or a Binary Large Object (BLOB). But the problem of managing the document size and slow searches and retrievals cannot be avoided.

Apache Xindice comes out a winner in document order preservation. That was precisely the reason for which it was designed. It supports collections of documents and searches and retrieves from an entire collection of documents when a query is executed.

### 4.1.6 Joins.

Join operations on content (or values) have been seen as a critical aspect that early XML query languages did not possess; particularly, data-centric applications that require a combination of data, based on values. The difficulties, bottlenecks, and challenges posed by joins are well-known from relational database systems.

Join operations are critical for relational database systems where the data is normalized in order to store and retrieve while achieving good performance. XML data stored in an RDBMS in a decomposed method can utilize the join operations of the RDBMS with which it interacts. MS SQL Server is no exception to this.

Apache Xindice on the other hand provides implicit join operations with its indexing facility that helps join documents that are logically connected, i.e., documents belonging to the same collection.

JAXB, unlike the other two does not provide an explicit join operation. The framework allows binding rules to be defined by the application processing the XML data. These binding rules may include any dependencies which can be translated into join operations on its RDBMS counterpart.

### 4.1.7 Containment and full-text search.

Containment and full-text search are elementary operations when querying XML. The problem and its intrinsic difficulties are well-known from other application domains.

Containment and full-text searches are supported by RDBMSes that support full-text searches on their data columns. For this purpose, XML data needs to be stored as a BLOB or CLOB. However, in such cases, the cost of searches cannot be avoided.

Apache Xindice provides indexing of element and attribute values in addition to the ability to manage and return aggregated document fragments. This helps in performing broad queries across a set of documents and speeds up the process just as a full-text search engine would do.

JAXB has no explicit full-text search functionality built into it. The application using it will have to handle any such requests.

The guidelines, described above, were followed to assess the performance of the software program. The software program for this comparative study was developed in Java 2.0 as it prevents biasing toward specific machine architectures, operating systems or environments.

## 4.2 Performance

Analysis of the time-complexity, space-complexity, complexity of interface program code, and ease of programming were conducted. To measure performance, the results were plotted as functions of input data (size) and complexity.

Results are produced with different values for the input parameters. The performance of the new implementation can be evaluated by the graphs produced. The graphs are plotted using Microsoft Excel spreadsheet software by inserting the results obtained from the simulation into a spreadsheet. The x-axis of the plot indicates the technology used namely JAXB, NXD and RDBMS. The y-axis of the plot indicates the time and space complexity as a single document and a collection of documents. The plot also contains the values of all the parameters used in running the simulation.

### 4.2.1 Execution Time

Figure 12 shows the average time variation with the various queries executed using the three technologies Java Architecture for XML Binding (JAXB), Apache Xindice Native XML Database (NXD) and Microsoft SQL Server 2000 SP3 (RDBMS).

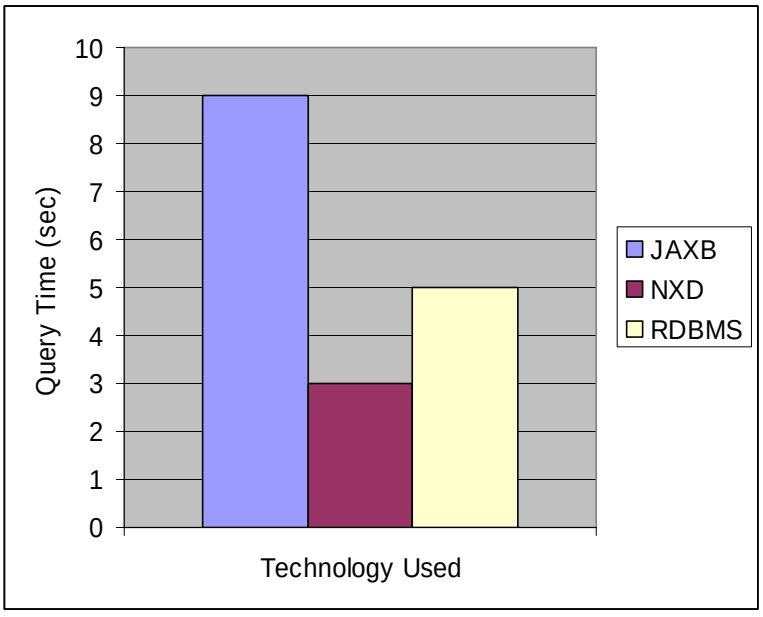


Figure 12: Query Execution Time complexity in various technologies used. Graph data based on Table II, Appendix C.

The query processing time was highest on JAXB as the content tree had to be loaded. The results in this case are slightly deceptive. Each time a search string was queried, a new instance of the application was run and hence the total time to execute the query included the time to create the JAXB content tree classes by the framework. But, by the results of Table II, Appendix C, it is clear that the shortest query time is using the JAXB classes provided all queries are made when the objects are loaded in memory, which is not mostly the case in enterprise applications.

The query processing time with NXD was faster than the rest of the technologies. RDBMS had an average performance.

### 4.2.2 Document size complexity

Figure 13 shows the document size complexity variation in the three technologies with for a single XML document.

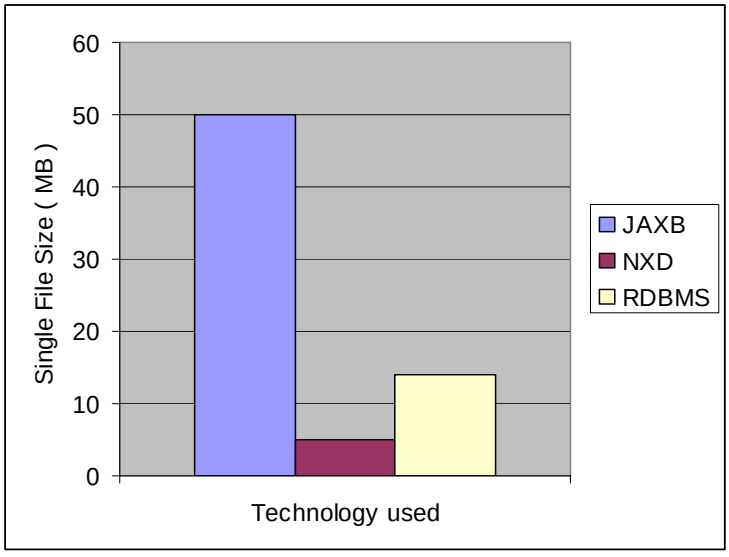


Figure 13: Space complexity in managing a single document. Data based on Table II, Appendix C.

MS SQL Server (RDBMS) had a mean performance in storing a single large XML document of size 14MB. This was made possible by extending the functionality of the stored procedure that loads the xml document into memory. For further scalability custom extension to the stored procedure is the only option.

Apache Xindice (NXD) on the other hand had a very poor capacity to store large documents. It cannot store and manage a single document that can be treated as a set of mini documents. It was not designed for the purpose. It performs fairly well when it comes to managing collections of small documents as shown in Figure 14.

Java Objects are practically not affected by the size of the document either singly or collectively as JAXB adheres to a single schema and manages memory more efficiently than the equivalent DOM structures. Hence, the size of the files that can be handled can vary with the available system resources.

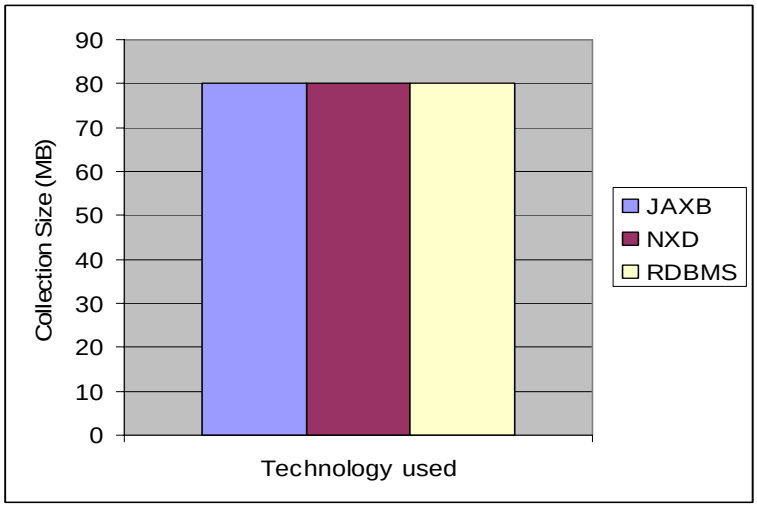


Figure 14: Space complexity of the technologies in managing a collection of documents. Graph Data based on Table II, Appendix C.

## 4.3 Performance of Queries

### 4.3.1 Insert Operations

The performance of inserts is shown in Figure 15. Calling get and set methods on Java Objects is much easier than inserting or updating nodes in a conventional DOM representation. As such, the performance of a single insert statement, with the content tree objects already loaded into memory is extremely good as shown in Figure 15.

RDBMS is also a close competitor in this regard but its performance suffers owing to the translation from XML to a relational schema for each operation.

XML Databases on the other hand are somewhat affected in their performance in inserting physically into a document. Though most NXD's make use of XML: DB API, which provide support for faster queries, they cannot beat the performance of RDBMS and Java Objects for individual inserts. Figure 15 shows the performance of the three technologies based on the average time (logarithmic scale) for a single insert.

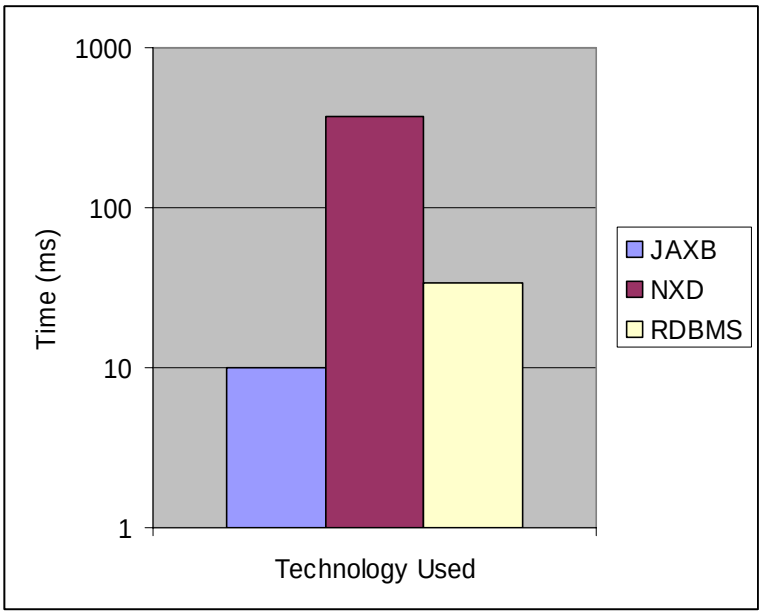


Figure 15: Performance comparisons of JAXB, NXD and RDBMS for individual insert statements

### 4.3.2 Select Operations

The performance of select statements is shown in Figure 16. Select or retrieval performance is similar for all the three technologies. As discussed in section 4.3.1., calling get and set methods on Java Objects is much easier than inserting or updating nodes in a conventional DOM representation. As such, the performance of select statements, with the content tree objects already loaded into memory is extremely good as shown in Figure 16.

RDBMS is also a close competitor in this regard but its performance suffers owing to the translation from XML to a relational schema for each operation.

XML Databases perform well in retrievals as they provide support for indexing on elements and attributes of the XML document to enable faster searches and retrievals.

Hence the overall performance in individual retrievals is close in all the three technologies. Figure 16 shows the performance of the three technologies based on the average time for a single retrieval.

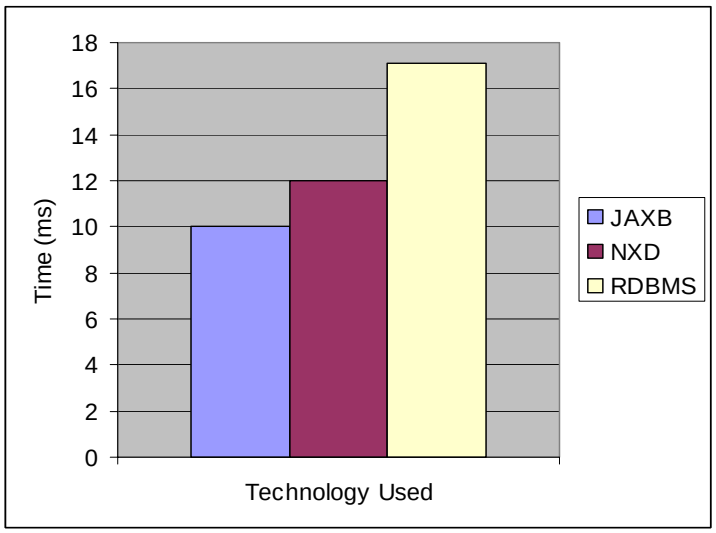


Figure 16: Performance comparisons of JAXB, NXD and RDBMS for individual select statements

### 4.3.3 Update Operations

The performance of updates is very similar to inserts, in that Java Objects and RDBMS technologies have an advantage with the performance as a measure of query execution time of an update.

Though Native XML Databases use support of third-party API (XML:DB in this context), to process an update at a faster rate by updating the document in place, they cannot achieve the performance of Java Objects or relational databases. This is similar to the problem that file systems encounter.

Figure 17 shows the performance of the three technologies for individual update queries with the average query execution time per query in ms (logarithmic scale) and the technology used.

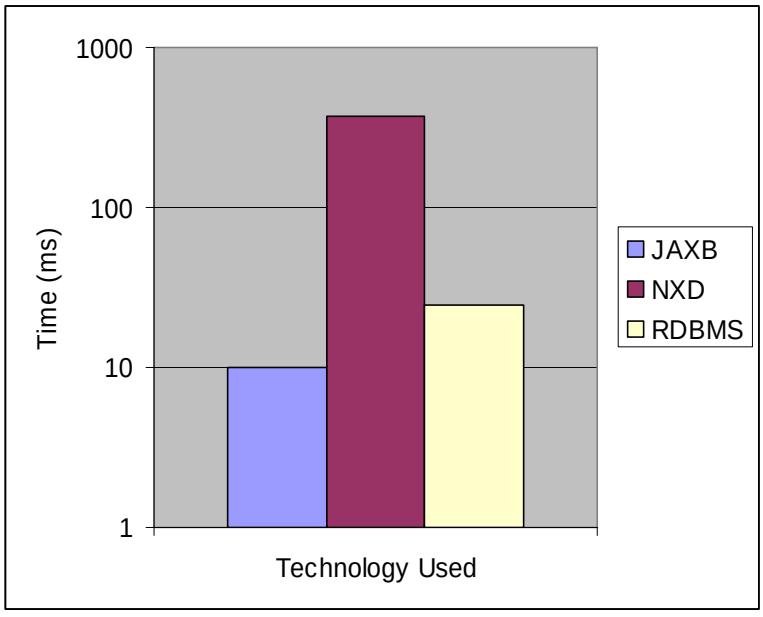


Figure 17: Performance comparisons of JAXB, NXD and RDBMS for individual update statements

### 4.3.4 Delete Operations

The performance of deletes is surprisingly very different from the performance of all the above query types, especially for Native XML databases. Java Objects perform well as with any other query. This is because of object manipulation as opposed to storage manipulation.

RDBMS performs better in case of deletes, than the other queries. This can be attributed to the regular delete process employed by the RDBMS. Though this requires translation, evaluation of the delete condition more often than not, requires lesser number of manipulations than its insert, update or select counterpart.

In the case of Native XML databases, the performance suffers heavily as the document management system or the Native DBMS actually deletes the content fragment from the document every time a delete operation is called. Figure 18 shows the performance of the three technologies for individual delete queries with the average query execution time per query in ms (logarithmic scale) and the technology used.

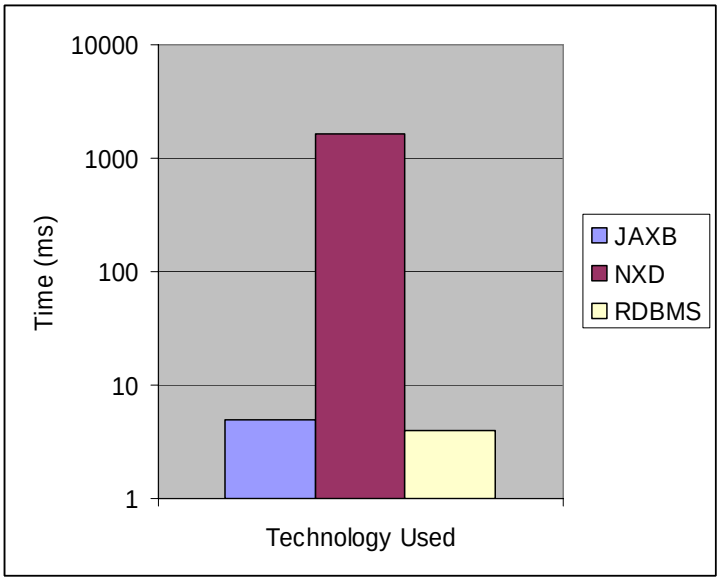


Figure 18: Performance comparisons of JAXB, NXD and RDBMS for individual delete statements

## 4.4 Comparison of features

Section 4.3.1 shows the observations of this study. These provide a high level overview of the advantages and disadvantages of one method over the other.

### 4.4.1 Storage and retrieval in RDBMS

- Transferring the data in an XML document to a relational database requires creating an XML schema and mapping to a database schema. Schemaless documents cannot be mapped to a relational database and the support of some mapping tool is required in order to use an RDBMS.
- The syntax of the OPENXML and FOR XML queries is not very easily comprehensible.
- The XML document used in queries needs to be declared as text variable in the SQL used in the queries. There is no support currently to redirect the document from a file.
- Generic use of XML templates over HTTP cannot be done using files.
- File redirection is supported only on proprietary languages supported by the platform.

### 4.4.2 Storage and retrieval in NXD

- The Apache Xindice 1.1b4 Native XML Database does not support valid XML documents. Hence, there is no mapping of schema required

- Xindice is not standalone. It needs a Web Application Server such as Apache Jakarta Tomcat or Jetty to provide XML DBMS services.
- Xindice stores and indexes compressed XML documents in order to provide that data to the client application with very little server-side processing overhead.
- It cannot store and manage single monster sized documents.
- Since Xindice runs under an application server, the database configuration is located in the Xindice sub-directory of the web application directory of the server.
- Schemaless or non-valid XML data can be stored and processed by Xindice.

### 4.4.3 Storage and retrieval using JAXB

- JAXB requires a well-formed and valid XML document to work with.
- XML documents without a conforming W3C Schema cannot be processed.
- The JAXB framework generates a collection of classes for every schema that is handled.
- It generates well-formed XML output during marshalling.
- Marshalling does not validate XML. Since the XML data has to be valid in order to unmarshall or the content tree generated from the ObjectFactory from Objects is validated by default, the marshaller does not perform validation.
- JAXB can work with multiple schemas at a time
- Data type conversions or casts can be performed by defining binding customizations in the binding file.
- JAXB sets no limit on the size of the XML document it can process.

## 4.5 Discussion

XML-enabled relational databases contain extensions for transferring data between XML documents and their own data structures. They can be used by data-centric applications. The difference between XML-enabled and native storage is that XML-enabled storage uses schema-specific structures that must be mapped to the XML document at design time. Native XML storage uses generic structures that can contain any XML document.

Native XML databases can preserve physical structure (entity usage, CDATA sections, etc.) as well as comments, processing instructions etc. While XML-enabled databases can do this in theory, this is generally not done in practice.

The only interface to the data in native XML databases is XML and related technologies, such as XPath, the DOM, or an XML-specific API. XML-enabled databases, on the other hand, offer direct access to the data, such as through ODBC.

“Wrappers” are systems that treat XML documents as a source of relational data. (The term comes from federated database systems, where a *wrapper* is a component that "wraps" a source system so its data uses the model (usually relational) of a target system. [3]) That is, with wrappers, XML data is treated as relational data, while with XML-enabled databases, relational data is treated as XML data. Wrappers typically implement an SQL query engine, use an object-relational or table-based mapping, and work only with data-centric documents.

MS SQL Server 2000 SP3 can be classified as a wrapper and also an XML-enabled database. SQL Server can be employed in many situations. One common situation is to use when XML document needs to be included in a heterogeneous join

(where a select statement joins data from different systems [1]). Another use can be as an XML editor. For those developers who are used to working with data in a tabular format, SQL Server comes in handy. The documents may later be converted into XML.

JAXB or any application that converts XML into Java Objects can be used in various situations such as:

- To access configuration values from a properties file stored in XML format
- To develop a tool that can create or modify a configuration properties file represented in XML format.
- To validate data input by a user; for example, from a form presented in a Web browser, where the form data is mapped to an XML document. In such cases, JAXB provides the capability to validate the accuracy of the data using the validation constraints of a schema that describes the data collected from the form.
- To bind an XML document into a Java representation, update the content via Java interfaces, validate these changes against the constraints within the original schema, and then write the updated Java representation back to an XML document.

## 4.6 Conclusion

### 4.6.1 Strengths

RDBMS provides good scalability, performance, reliability and data access, features that are supported by the DBMS

Native XML databases can be accessed using CORBA, Java API, XML:RPC and other API making them portable over the web. It provides collection management, and preserves structure and order.

Java Objects provide ease of implementation, so much so that the end user is not expected to know anything about XML. Java objects provide abstract data type support.

### 4.6.2 Weak points

RDBMS requires mapping of semi-structured XML data onto relational structured data storage. Ordering of elements is required. In order to ensure correct mapping multiple tables and multiple joins are required which lead to bad performance.

Native XML databases suffer from performance in ‘multiple queries to a single document’ situations. There is no unique or standard implementation of the NXD. The utilities and tools provided are proprietary contributions of the vendor.

Storage of XML in Java Objects results in document factorization. Besides, Java Objects are tied up to a schema (which, is required for processing) thereby restricting the processing only to well-formed XML.

# CHAPTER V

# SUMMARY AND FUTURE WORK

This chapter gives a brief summary of this thesis and also discusses some of the future work that can be done in this area.

## 5.1 Summary

A detailed comparative study on storage of XML using Relational DBMS, Native XML DBMS and processing into Java Objects using JAXB was conducted. The data models such as relational, hierarchical, document-driven were used as inputs to the study. There is no single tool that can manage all the aspects of XML data used in an application. Each technology provides interestingly unique features.

There is a tremendous amount of research and development in progress, in the development of tools and technologies to use XML. It can be safely predicted that all the technologies will finally merge into one standard method of storage of XML that will incorporate all the features such as, faster searches, full-text searches, with an ability to maintain a collection of documents and preserve their order, ability to query and store or retrieve over the network using protocols such as HTTP, SOAP etc., provide integral support for casting of elements, support for processing valid and non-valid XML

documents, all in a single tool and also achieve platform independence to be called a truly portable tool for portable data.

## 5.2 Future work

As XML continues to evolve, its usage will grow tremendously in all facets of application development. Future work of this study will incorporate testing across various platforms along with other emerging technologies such as XML servers and Content Management Systems and delve into the intricacies of XML Query Engines, their efficiency, and XML Data Binding and perform a comparative analysis of the technologies irrespective of their platform of test or implementation.

REFERENCES


1. Altova XML Suite. Altova GmbH Corporation. http://www.altova.com. Last accessed April 18, 2005.
2. Armstrong, Eric. XML Glossary. Working with XML. Sun Microsystems. http://java.sun.com/xml/jaxp/dist/1.0.1/docs/tutorial/glossary.html#D Last Updated July 13, 2000. Last accessed April 18, 2005.
3. Bourret, Ronald. XML and Databases XML Database Links. http://www.rpbourret.com/xml/XMLAndDatabases.htm Last updated November 2003. Last accessed April 18, 2005.
4. Champion Michael, Storing XML in Databases. Business Integration Journal, October 2001. Last accessed April 18, 2005.
5. Cheng, Josephine and Xu, Jane. IBM DB2 XML Extender: An End to End Solution for Storing and Retrieving XML Documents. International Conference on Data Engineering 2000.
6. Cincom Corporate, Choosing an XML Database Solution. White Paper. http://tiger.cincom.com/PDFs/TOTALXML_Choosing_XML_Database.pdf Last updated January 1, 2004. Last accessed April 18, 2005.
7. Clark, James. Multi-format Schema converter based on RELAX-NG. Trang. http://thaiopensource.com/relaxng/trang.html Last accessed April 18, 2005.

8. Clark, James. DTDinst. Program for converting XML document to its DTD. DTDInst http://www.thaiopensource.com/relaxng/dtdinst/ Last accessed April 18, 2005.

9. Connolly Thomas, Begg Carolyn. Database Systems. A Practical Approach to Design Implementation, and Management. 3rd edition, Addison-Wesley 2002. pp 1006 -1007.

10. Cunningham & Cunningham, Inc. Hierachical Databases. Article, January 4, 2004. http://c2.com/cgi/wiki?HierarchicalDatabase Last updated January 2004. Last accessed April 18, 2005.

11. Dayen, Igor. Storing XML in Relational Databases. June 2001. http://www.xml.com/pub/a/2001/06/20/databases.html?page=1 Last updated June 2004. Last accessed April 18, 2005.

12. Dejesus, Edmund. XML Enters the DBMS Arena. Computer World. October 2000. http://www.computerworld.com/cwi/story/0,1199,NAV63_STO53026,00.htm Last updated January 2001. Last accessed April 18, 2005.

13. Documentation. Apache Xindice 1.1b4. Apache Software Foundation. http://xml.apache.org/xindice Last accessed April 18, 2005

14. DuCharme, Bob. Documents vs. Data, Schemas vs. Schemas. XML 2004 Conference Proceedings. Washington DC. November 16, 2004.

15. Emerick, Jerry. Managing XML Data Storage. ACM Crossroads 8.4, Summer 2002. http://www.acm.org/crossroads/xrds8-4/XML_RDBMS.html. Last updated August 2002. Last accessed April 18, 2005.

16. Fordin, Scott. Java Architecture for XML Binding Executive Summary. Java and XML Technologies. Sun Microsystems. July 2003. http://www.sun.com/software/xml/developers/jaxb/index.xml Last accessed April 19, 2005.

17. Goldfarb, Charles F., Prescod, Paul. XML and databases. XML Handbook 4th Edition. Prentice Hall 2002. pp. 430-488.

18. Gupta Samudra. The Mysteries of Business Objects. Article. http://javaboutique.internet.com/tutorials/businessObject/ . Last Updated July 21, 2004. Last accessed April 18, 2005.

19. Hoque Reaz, XML for Real Programmers, Morgan Kaufmann, 2000. pp 379-442.

20. Jackson Laboratory. Glossary. Mouse Genome Informatics. http://www.informatics.jax.org/mgihome/other/glossary.shtml#d Last Updated 2004. Last accessed April 18, 2005.

21. John, Murray Crosswell. Reading an XML file inside a Stored Procedure. Experts Exchange.com http://www.experts-exchange.com/Databases/Microsoft_SQL_Server/Q_20670044.html October 21, 2003. Last accessed April 18, 2005

22. Laux, Andreas., Martin, Lars. API for XML Databases. XML:DB Working Draft 2000. Last accessed April 18, 2005.

23. Layman Andrew. XML syntax recommendation for serializing graphs of data. December 2, 1998. World Wide Web Consortium. http://www.w3.org/TandS/QL/QL98/pp/microsoft-serializing.html. Last accessed April 18, 2005.

24. Lewis, William. Pillar of the Community. Intelligent Enterprise. August 2000 http://www.intelligententerprise.com/000818/feat1.shtml. Last updated February 2001. Last accessed April 18, 2005.

25. Maruyama Hiroshi, Tamura Kent, Uramoto Naohiko, XML and Java, Developing Web Applications. Addison-Wesley 1999. Pp185-228.

26. McGrath, Sean. Xml-dev posting, June 3, 2004. Available at http://lists.xml.org/archives/xml-dev/200406/msg00022.html Last accessed April 18, 2005.

27. McLaughlin, Brent. A survey of existing XML output options. IBM developerWorks. March 26, 2003. http://www-128.ibm.com/developerworks/xml/library/x-tipbigdoc.html Last accessed April 19, 2005.

28. MSDN library. IIS Virtual Directory Management for SQL Server. Microsoft Corporation. http://msdn.microsoft.com/library/default.asp?url=/library/en-us/sqlxml3/htm/intro_9ol4.asp Last accessed April 18, 2005.

29. Ort, Ed., Mehta, Bhakti. Java Architecture for XML Binding. Sun Microsystems. http://java.sun.com/developer/technicalArticles/WebServices/jaxb/index.html March 2003. Last accessed April 18, 2005.

30. Policht, Marcin. XML and SQL 2000. Database Journal.com http://www.databasejournal.com/features/mssql/article.php/10894_2235451_2 August 12, 2003. Last accessed April 18, 2005.

31. Raggett Dave. HTML 3.2 Reference Specifications. World Wide Web Consortium. January 14, 1997. http://www.w3.org/TR/dtd Last accessed April 18, 2005.

32. Refsnes Data. W3Schools. Introduction to XML http://www.w3schools.com/xml/xml_whatis.asp Last Updated January 2004. Last accessed April 18, 2005.

33. Schaffner, Brian. Storing large XML documents in Relational Databases. Article September 29, 2003 http://builder.com.com/5100-31-5075709.html Last updated January 2004. Last accessed April 18, 2005.

34. Schmidt Albrecht, Processing XML in Database Systems. Ph.D. Thesis, Universiteit van Amsterdam, Amsterdam, The Netherlands, November 2002.

35. Schmidt Albrecht, Waas Florian, Kersten Martin, Manolescu Ioana, Carey Michael J., Busse Ralph. The XML Benchmark Project. Technical Report INS-R0103, CWI, Amsterdam, The Netherlands, April 2001.

36. Shanmugasundaram Jayavel, Shekita Eugene, Kiernan Jerry, Krishnamurthy Rajasekar, Viglas Efstratios, Naughton Jefferey, Tatarinov Igor. A General Technique for Querying XML Documents using a Relational Database System. Department of Computer Sciences, University of Wisconsin-Madison. 2000.

37. Shapiro, Jefferey R. SQL Server and the Internet. SQL Server 2000. The Complete Reference. Osborne/McGraw-Hill. 2001. pp. 800-810.

38. Suciu, Dan. XML Data Repository. Computer Science and Engineering Department. University of Washington. http://www.cs.washington.edu/research/xmldatasets/ . Last modified Nov 12, 2002. Last accessed April 18, 2005.

39. Sun Developer Network Site, Java Technology and XML-Part 3: Performance Improvement Tips. Technical Articles and Tips in XML. June 2004.

http://java.sun.com/developer/technicalArticles/xml/JavaTechandXML_part3/ Last updated June 2004. Last accessed April 18, 2005.

40. W3C Architecture Domain. The Extensible Style Sheet Language Family. World Wide Web Consortium 2004. http://www.w3.org/Style/XSL/ Last accessed April 18, 2005.

41. XML specification. Extensible Markup Language 1.0. W3C Recommendation February 4, 2004. http://www.w3.org/TR/REC-xml/ Last accessed April 19, 2005.

# APPENDICES

# APPENDIX A

## ABBREVIATIONS

| | |
|---|---|
| API | Application Programming Interface |
| DBMS | Database management system |
| JAXB | Java Architecture for XML Binding |
| NXD | Native XML Database |
| XML | eXtensible Markup Language |
| XSL | eXtensible Style Language |
| XSLT | eXtensible Style Language Transformation |

# APPENDIX B

## TRADEMARK INFORMATION

| | |
|---|---|
| Excel | A registered trademark of Microsoft Corporation. |
| Java Runtime Environment 1.4.2_06 | A registered trademark of Sun Microsystems Inc. |
| Windows XP Professional | A registered trademark of Microsoft Corporation. |

## APPENDIX C

This appendix contains the results generated by the three separate technologies JAXB, NXD and RDBMS.

| Query | Nesting level | Technology | Time (ms ) | Size (KB) | Rows |
|---|---|---|---|---|---|
| Insert | 2 | RDBMS | 10 | 25 | 10 |
| Insert | 3 | RDBMS | 62 | 120 | 60 |
| Insert | 4 | RDBMS | 113 | 146 | 40 |
| Update | 2 | RDBMS | 8 | 25 | 40 |
| Update | 4 | RDBMS | 185 | 146 | 30 |
| Update | 3 | RDBMS | 172 | 120 | 70 |
| Delete | 2 | RDBMS | 47 | 25 | 100 |
| Delete | 3 | RDBMS | 80 | 120 | 50 |
| Delete | 4 | RDBMS | 180 | 146 | 40 |
| Select | 3 | RDBMS | 180 | 130 | 60 |
| Select | 4 | RDBMS | 212 | 146 | 40 |
| Select | 3 | NXD | 2063 | 273 | 520 |
| Select | 4 | NXD | 270 | 240 | 80 |
| Insert | 3 | NXD | 3015 | 273 | - |
| Insert | 4 | NXD | 1213 | 146 | - |
| Delete | 3 | NXD | 766 | 273 | - |
| Delete | 3 | NXD | 198 | 273 | 40 |
| Update | 3 | NXD | 167 | 273 | 70 |
| Update | 4 | NXD | 1716 | 300 | 703 |
| Insert | 3 | JAXB | 10000 | 135 | * |
| Insert | 4 | JAXB | 9000 | 1022 | * |
| Update | 3 | JAXB | 8500 | 1022 | * |
| Select | 3 | JAXB | 8800 | 1022 | * |
| Select | 3 | JAXB | 9100 | 300 | * |

Table II: Results of query processing comparison

Table II shows the variation of query performance with granularity of XML documents and the technologies studied.

- Apache XML database processes documents and collections of documents at a time as opposed to an RDBMS row-wise query. Retrieval however returns the result set in rows.

* The XML documents were processed individually by the JAXB compiler into Java objects which cannot be compared to rows as in an RDBMS

# APPENDIX D

The following are the programs generated for the three different technologies that store and use the XML data. XML data is loaded into each of the methods using

## CODE LISTING

```
/*********************************************************************
Student Name:    UMA CHINMAYEE NIMRAL
Thesis Title:    EFFICIENT STORAGE OF XML – A COMPARATIVE STUDY
Advisor:         DR.BLAYNE E. MAYFIELD
Estimated Times: DESIGNING: 40-50 hrs
                 IMPLEMENTATION: 75-80 hrs
                 TESTING: 150-200 hrs
*********************************************************************/
// Stored Procedure u_xmlfile_preparedocument

CREATE PROCEDURE dbo.u_xmlfile_preparedocument @hdoc Integer Output, @xmlfile
varchar(1000) = NULL
AS
--
--    Description:
--    Utility procedure to invoke sp_xml_preparedocument from a file rather than text
--    Credits:
--    Joseph Gama
(http://msdn.microsoft.com/library/default.asp?url=/library/enus/dnsqlpro03/html/sp03g8.a
sp)
--        Umachandar Jayachandran (http://www.sqlxml.org/faqs.aspx?faq=42)
--    24-Nov-2003 Murray Crosswell
--
--
Begin

SET NOCOUNT ON

Declare @i int,
        @size int,
        @blocks int,
        @varcnt int,
        @qm char(1),
        @bperv integer,
        @cmdsrc varchar(8000),
        @cmdsrc2 varchar(8000),
        @cmdsrc3 varchar(8000),
        @CRLF Varchar(2)

Set @CRLF = Char(13) + Char(10)

create table #filedetails
(altname varchar(30),size int,createdate varchar(32),
 createtime varchar(32),lastwrittendt varchar(30),
 lastwrittentime varchar(32),lastaccessdt varchar(30),
 lastaccesstime varchar(32),attributes int)

insert into #filedetails exec master..xp_getfiledetails @xmlfile

set @size=(select size from #filedetails)
drop table #filedetails
```

```
Set @qm = char(39)
Set @blocks=@size/4000+1
Set @varcnt=(@blocks-1)/@bperv+1

Set @cmdsrc = 'Declare @cmdDS varchar(8000), '
Set @cmdsrc = @cmdsrc + '@cmdES varchar(8000), '
Set @i=1
While @i<=@varcnt
  Begin
  Set @cmdsrc = @cmdsrc + '@cmdD' + Convert(Varchar(9),@i) + ' Varchar(8000), '
  Set @cmdsrc = @cmdsrc + '@cmdR' + Convert(Varchar(9),@i) + ' Varchar(8000), '
  Set @cmdsrc = @cmdsrc + '@cmdF' + Convert(Varchar(9),@i) + ' Varchar(8000), '
  Set @cmdsrc = @cmdsrc + '@cmdE' + Convert(Varchar(9),@i) + ' Varchar(8000), '
  Set @i=@i+1
  End
Set @cmdsrc = @cmdsrc + '@cmdDF varchar(8000), '
Set @cmdsrc = @cmdsrc + '@cmdEF varchar(8000), '
Set @cmdsrc = @cmdsrc + '@i int, @blocks int'

Set @cmdsrc = @cmdsrc + ' SET @blocks=' + Convert(Varchar(9),@blocks)
Set @cmdsrc = @cmdsrc + ' SET @cmdDS='+@qm+'DECLARE @fso int, @fl int, @hr int, @qm
char(1), @vp varchar(8000), @vs varchar(8000), '+@qm

Set @cmdsrc = @cmdsrc + ' SET @cmdDF='+@qm+' Set @qm = char(39)'+@qm
Set @cmdsrc = @cmdsrc + ' SET @cmdDF=@cmdDF+'+@qm+' Set @vp = '+@qm+@qm+'DECLARE @Handle
int EXEC sp_xml_preparedocument @Handle OUT, '+@qm+@qm+'+@qm'+@qm
Set @cmdsrc = @cmdsrc + ' SET @cmdDF=@cmdDF+'+@qm+' Set @vs = @qm+'+@qm+@qm+' DECLARE
he_cur CURSOR GLOBAL FOR SELECT @Handle'+@qm+@qm+@qm
Set @cmdsrc = @cmdsrc + ' SET @cmdDF=@cmdDF+'+@qm+' EXEC @hr = sp_OACreate
'+@qm+@qm+'Scripting.FileSystemObject'+@qm+@qm+', @fso OUT '+@qm
Set @cmdsrc = @cmdsrc + ' SET @cmdDF=@cmdDF+'+@qm+'exec sp_oamethod @fso,
'+@qm+@qm+'opentextfile'+@qm+@qm+', @fl out, '+@qm+@qm+@xmlfile+@qm+@qm+', 1 '+@qm
Set @i=1
While @i<=@varcnt
  Begin
  Set @cmdsrc = @cmdsrc + ' SET @cmdD' + Convert(Varchar(9),@i) + '='+@qm+@qm
  Set @cmdsrc = @cmdsrc + ' SET @cmdR' + Convert(Varchar(9),@i) + '='+@qm+@qm
  Set @cmdsrc = @cmdsrc + ' SET @cmdF' + Convert(Varchar(9),@i) + '='+@qm+@qm
  Set @cmdsrc = @cmdsrc + ' SET @cmdE' + Convert(Varchar(9),@i) + '='+@qm+@qm
  Set @i=@i+1
  End
Set @cmdsrc = @cmdsrc + ' SET @i=1'
Set @cmdsrc = @cmdsrc + ' WHILE @i<=@blocks'
Set @cmdsrc = @cmdsrc + ' BEGIN'
Set @cmdsrc2 = ''
Set @i=1
While @i<=@varcnt
  Begin
  If @i <> 1 Set @cmdsrc2 = @cmdsrc2 + ' ELSE'
  If @i <> @varcnt Set @cmdsrc2 = @cmdsrc2 + ' IF @i < ' +
Convert(Varchar(9),@i*@bperv+1)
  If @varcnt > 1 Set @cmdsrc2 = @cmdsrc2 + ' BEGIN'
  Set @cmdsrc2 = @cmdsrc2 + ' SET @cmdD' + Convert(Varchar(9),@i) + '=@cmdD' +
Convert(Varchar(9),@i) + '+'+@qm+'@v'+@qm+'+CONVERT(VARCHAR(9),@i)+'+@qm+' varchar(8000),
'+@qm
  Set @cmdsrc2 = @cmdsrc2 + ' SET @cmdR' + Convert(Varchar(9),@i) + '=@cmdR' +
Convert(Varchar(9),@i) + '+'+@qm+'exec @hr=sp_oamethod
@fl,'+@qm+@qm+'read'+@qm+@qm+','+@qm+'+'+@qm+'@v'+@qm+'+CONVERT(VARCHAR(9),@i)+'+@qm+'
out,4000 '+@qm
  Set @cmdsrc2 = @cmdsrc2 + ' SET @cmdF' + Convert(Varchar(9),@i) + '=@cmdF' +
Convert(Varchar(9),@i) + '+'+@qm+'Set
@v'+@qm+'+CONVERT(VARCHAR(9),@i)+'+@qm+'=Replace(@v'+@qm+'+CONVERT(VARCHAR(9),@i)+'+@qm+'
,@qm,@qm+@qm) '+@qm
  Set @cmdsrc2 = @cmdsrc2 + ' SET @cmdE' + Convert(Varchar(9),@i) + '=@cmdE' +
Convert(Varchar(9),@i) + '+'+@qm+'@v'+@qm+'+CONVERT(VARCHAR(9),@i)+'+@qm+'+'+@qm
  If @varcnt > 1 Set @cmdsrc2 = @cmdsrc2 + ' END'
  Set @i=@i+1
  End
Set @cmdsrc3 = ' SET @i=@i+1'
Set @cmdsrc3 = @cmdsrc3 + ' END'
```

```
Set @cmdsrc3 = @cmdsrc3 + ' SET @cmdD' + Convert(Varchar(9),@varcnt) + '=LEFT(@cmdD' +
Convert(Varchar(9),@varcnt) + ',len(@cmdD' + Convert(Varchar(9),@varcnt) + ')-1)'

Set @cmdsrc3 = @cmdsrc3 + ' SET @cmdES='+@qm+'exec(@vp+'+@qm
Set @cmdsrc3 = @cmdsrc3 + ' SET @cmdEF='+@qm+'@vs) EXEC @hr = sp_OADestroy @fl EXEC @hr =
sp_OADestroy @fso'+@qm
Set @cmdsrc3 = @cmdsrc3 + ' exec( @cmdDS+'
Set @i=1
While @i<=@varcnt
  Begin
  Set @cmdsrc3 = @cmdsrc3 + '@cmdD' + Convert(Varchar(9),@i) + '+'
  Set @i=@i+1
  End
Set @cmdsrc3 = @cmdsrc3 + '@cmdDF+'
Set @i=1
While @i<=@varcnt
  Begin
  Set @cmdsrc3 = @cmdsrc3 + '@cmdR' + Convert(Varchar(9),@i) + '+'
  Set @i=@i+1
  End
Set @i=1
While @i<=@varcnt
  Begin
  Set @cmdsrc3 = @cmdsrc3 + '@cmdF' + Convert(Varchar(9),@i) + '+'
  Set @i=@i+1
  End
Set @cmdsrc3 = @cmdsrc3 + '@cmdES+'
Set @i=1
While @i<=@varcnt
  Begin
  Set @cmdsrc3 = @cmdsrc3 + '@cmdE' + Convert(Varchar(9),@i) + '+'
  Set @i=@i+1
  End
Set @cmdsrc3 = @cmdsrc3 + '@cmdEF )'

Execute (@cmdsrc+@cmdsrc2+@cmdsrc3)

OPEN GLOBAL he_cur
FETCH he_cur INTO @hdoc
DEALLOCATE GLOBAL he_cur

IF @hdoc IS NULL
    RAISERROR( 'Invalid Handle!', 16, 1 )

End
GO

////////////////// END STORED PROCEDURE ////////////////////////


import java.sql.*;
import java.io.*;

public class sqlXmlJava
{
       public sqlXmlJava() throws Exception
       {
              //get connection
              try
              {
                     Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");


//            DriverManager.registerDriver( new
com.micrsoft.jdbc.sqlserver.SQLServerDriver() );
              Connection connect = DriverManager.getConnection("jdbc:odbc:thdata", "sa",
"polaris");
              Statement st = connect.createStatement();

              BufferedReader stdin = new BufferedReader( new InputStreamReader(
System.in ) );
```

```
                if( connect != null )
                {
                        System.out.println();
                        System.out.println( "Successfully connected " );
                        System.out.println();


                        System.out.println(" Enter Sql query string " );
                        String input = stdin.readLine();


                        st.executeUpdate("Use test");

                        ResultSet rs = st.executeQuery(input);

                }
           }//end try
           catch( Exception e )
                {
                        System.err.println(" Exception :" + e.getMessage());
                }
        }//  constructor


        public static void main( String[] args ) throws Exception
        {
                sqlXmlJava test = new sqlXmlJava();

        }

}

//////////////////////
Sample SQL files used

Insert.sql
-----------------------------------
DECLARE @iDoc int

EXEC u_xmlfile_preparedocument @iDoc OUTPUT, 'C:\thesis\reed.xml'

INSERT INTO course
SELECT * FROM OPENXML (@iDoc, '/root/course', 3)
WITH course

INSERT INTO time
SELECT * FROM OPENXML (@iDoc, '/root/course/time', 3 )
WITH time

INSERT INTO place
SELECT * FROM OPENXML( @iDoc, '/root/course/place', 3 )
WITH place

EXEC sp_xml_removedocument @iDoc


Update.sql
-----------------------------------

DECLARE @iDoc int

EXEC u_xmlfile_preparedocument @iDoc OUTPUT, 'C:\thesis\reed.xml'

UPDATE course
SET course.crse='H'+ c.crse
FROM OPENXML( @iDoc, '/root/course[subj="HIST"]',3 )
WITH  course c WHERE course.subj=c.subj

EXEC sp_xml_removedocument @iDoc
```

```
Delete.sql
-----------------------------------------

DECLARE @iDoc int

EXEC u_xmlfile_preparedocument @iDoc OUTPUT, 'C:\thesis\reed.xml'

DELETE
FROM course
WHERE crse IN
        (SELECT crse FROM course, (SELECT reg_num FROM OPENXML(@iDoc,
'/root/course[instructor=""]',3)
                        WITH course )t
        WHERE course.reg_num = t.reg_num )

EXEC sp_xml_removedocument @iDoc
```

```
TECHNOLOGY 2 – Apache Xindice 1.1b4 Native XML Database
------------------------------------------------------------


// AbstractExample.java

package org.apache.xindice.examples;


import org.apache.xindice.util.XindiceException;
import org.xmldb.api.DatabaseManager;
import org.xmldb.api.base.Collection;
import org.xmldb.api.base.Database;
import org.xmldb.api.base.XMLDBException;

/**
 * Simple XML:DB API Example
 */
public abstract class AbstractExample {

    private static boolean alreadyInitialized = false;

    protected static void ensureInitialized()
        throws ClassNotFoundException, InstantiationException, IllegalAccessException,
               XMLDBException {

        if (alreadyInitialized) {
            return;
        }

        String driver = "org.apache.xindice.client.xmldb.DatabaseImpl";
        Class driverClass = Class.forName(driver);
        Database database = (Database) driverClass.newInstance();
        DatabaseManager.registerDatabase(database);
        alreadyInitialized = true;
    }

    protected static Collection getCollection(String collectionUri)
        throws ClassNotFoundException, InstantiationException, IllegalAccessException,
               XMLDBException, XindiceException {

        ensureInitialized();
        Collection collection = DatabaseManager.getCollection(collectionUri);
        if (collection == null)
         {
            throw new XindiceException("DatabaseManager.getCollection(" + collectionUri +
") returned null.");
```

```
        }

        return collection;
    }
}

/// Test1.java
package org.apache.xindice.examples;


import org.xmldb.api.base.Collection;
import org.xmldb.api.base.Resource;
import org.xmldb.api.base.ResourceIterator;
import org.xmldb.api.base.ResourceSet;
import org.xmldb.api.base.XMLDBException;
import org.xmldb.api.modules.XPathQueryService;

/**
 * Simple XML:DB API example to query the database.
 */
public class Test1 extends AbstractExample {

    public static void main(String[] args) throws Exception {
        Collection collection = null;
        try {

            collection = getCollection("xmldb:xindice:///db/addressbook");

            String xpath = "//person[fname='John']";
            XPathQueryService service = (XPathQueryService)
collection.getService("XPathQueryService", "1.0");
            ResourceSet resourceSet = service.query(xpath);
            ResourceIterator resourceIterator = resourceSet.getIterator();

            while (resourceIterator.hasMoreResources()) {
                Resource resource = resourceIterator.nextResource();
                System.out.println((String) resource.getContent());
            }
        } catch (XMLDBException e) {
            System.err.println("XML:DB Exception occured " + e.errorCode + " " +
e.getMessage());
        } finally {
            if (collection != null) {
                collection.close();
            }
        }
    }
}

//List Collections.java
package org.apache.xindice.examples;


import org.xmldb.api.base.Collection;
import org.xmldb.api.base.XMLDBException;

/**
 * Simple XML:DB API example to list collections.
 */
public class ListCollections extends AbstractExample {

    public static void main(String[] args) throws Exception {
        Collection collection = null;
        try {

            collection = getCollection("xmldb:xindice://localhost:8888/db/");
            //collection = getCollection("xmldb:xindice-embed:///db/");

            String[] childCollections = collection.listChildCollections();
            System.out.println("Children of collection [" + collection.getName() + "]");
            for (int i = 0; i < childCollections.length; i++) {
```

```
                System.out.println("\t" + (i + 1) + ". " + childCollections[i]);
            }
        } catch (XMLDBException e) {
            System.err.println("XML:DB Exception occured " + e.errorCode + " " +
e.getMessage());
            e.printStackTrace();
        } catch (Exception e) {
            System.err.println("Exception occured " + e);
            e.printStackTrace();
        } finally {
            if (collection != null) {
                collection.close();
            }
        }
    }
}


//Retrieve Document.java
package org.apache.xindice.examples;

import org.xmldb.api.base.Collection;
import org.xmldb.api.base.XMLDBException;
import org.xmldb.api.modules.XMLResource;

/**
 * Simple XML:DB API Example to retrieve a document from the database.
 */
public class RetrieveDocument extends AbstractExample {

    public static void main(String[] args) throws Exception {
        Collection collection = null;
        try {

            collection = getCollection("xmldb:xindice:///db/addressbook");

            XMLResource xmlResource = (XMLResource) collection.getResource(args[0]);
            if (xmlResource != null) {
                System.out.println("Document " + args[0]);
                System.out.println(xmlResource.getContent());
            }
            else {
                System.out.println("Document not found");
            }
        } catch (XMLDBException e) {
            System.err.println("XML:DB Exception occured " + e.errorCode + " " +
e.getMessage());
        } finally {
            if (collection != null) {
                collection.close();
            }
        }
    }
}

// Xupdate.java
package org.apache.xindice.examples;


import org.xmldb.api.base.Collection;
import org.xmldb.api.base.XMLDBException;
import org.xmldb.api.modules.XUpdateQueryService;

/**
 * Simple XML:DB API example to update the database.
 */
public class XUpdate  extends AbstractExample {

    public static void main(String[] args) throws Exception {
        Collection collection = null;
```

```
        try {

            collection = getCollection("xmldb:xindice:///db/addressbook");

            String xupdate =
                "<xu:modifications version=\"1.0\""
                    + "      xmlns:xu=\"http://www.xmldb.org/xupdate\">"
                    + "   <xu:remove select=\"/person/phone[@type = 'home']\"/>"
                    + "   <xu:update select=\"/person/phone[@type = 'work']\">"
                    + "       480-300-3003"
                    + "   </xu:update>"
                    + "</xu:modifications>";

            XUpdateQueryService service = (XUpdateQueryService)
collection.getService("XUpdateQueryService", "1.0");
            long count = service.update(xupdate);
            System.out.println(count + " entries updated.");
        } catch (XMLDBException e) {
            System.err.println("XML:DB Exception occured " + e.errorCode + " " +
e.getMessage());
        } finally {
            if (collection != null) {
                collection.close();
            }
        }
    }
}



// Search for a document fragment
package org.apache.xindice.examples;

import org.apache.xindice.util.XindiceException;
import org.xmldb.api.DatabaseManager;

import org.xmldb.api.base.Collection;
import org.xmldb.api.base.Database;
import org.xmldb.api.base.XMLDBException;
import org.xmldb.api.base.Resource;
import org.xmldb.api.base.ResourceIterator;
import org.xmldb.api.base.ResourceSet;

import org.xmldb.api.modules.XPathQueryService;

import org.apache.xindice.Stopwatch;

public class searchDocumentFragment extends AbstractExample
{
        public static void main ( String args[] ) throws Exception
        {
                Collection collection = null;
                Collection col = null;
                String xpath = "";
                Stopwatch watch = new Stopwatch();

                //check if search string is not given
                if( args.length == 0 )
                {
                        System.out.println(" Illegal invocation : Usage [cmd] \"xpath
search string\" " );
                        System.exit(0);
                }
                xpath = args[0];

                try
                {
                        watch.start();
                         collection = getCollection("xmldb:xindice://localhost:8888/db/");

                        String[] childCollections = collection.listChildCollections();
```

```
                        System.out.println("Collection " + collection.getName() + " has " +
collection.getChildCollectionCount() +" children ");

                        for (int i = 0; i < childCollections.length; i++)
                        {
                                System.out.println("\t" + (i + 1) + ". " +
childCollections[i]);
                                try
                                {
                                        String colstring = "xmldb:xindice:///db/" +
childCollections[i];

                                        col = getCollection(colstring);
                                        XPathQueryService service = (XPathQueryService)
col.getService("XPathQueryService", "1.0");

                                        System.out.println (" the query string is : "+ xpath
);

                                        ResourceSet resourceSet = service.query(xpath);
                                        ResourceIterator resourceIterator =
resourceSet.getIterator();

                                        while (resourceIterator.hasMoreResources())
                                        {
                                                Resource resource =
resourceIterator.nextResource();
                                                System.out.println((String)
resource.getContent());
                                        }
                                }
                                catch (XMLDBException e)
                                {
                                        System.err.println("XML:DB Exception occured " +
e.errorCode + " " + e.getMessage());
                                }
                                finally
                                {
                                        if (col != null)
                                        {
                                              col.close();
                                        }
                                }
                                System.out.println ();

                        }//end for

                        watch.stop();
                }
                catch (XMLDBException e)
                {
                        System.err.println("XML:DB Exception occured " + e.errorCode + " "
+ e.getMessage());
                 }
                finally
                {
                        if (collection != null)
                        {
                            collection.close();
                        }
                }

                System.out.println(" Total time elapsed : "+ watch.elapsed() + "ms" );
        } //end main
}

// Delete Document
package org.apache.xindice.examples;

import org.xmldb.api.base.Collection;
import org.xmldb.api.base.Resource;
```

```
import org.xmldb.api.base.XMLDBException;

import org.apache.xindice.Stopwatch;

/**
 * Simple XML:DB API Example to delete a document from the database.
 */
public class DeleteDocument extends AbstractExample
{
    public static void main(String[] args) throws Exception
    {
        Collection collection = null;
          Stopwatch watch = new Stopwatch();
        try
          {

                if(args.length < 2 )
                {
                        System.out.println(" Incorrect call: Usage [cmd] [resourceid]
[collection] ");
                        System.exit(0);
                }
                String colstr = "xmldb:xindice:///db/" + args[1];

                watch.start();
             collection = getCollection(colstr);

                Resource document = collection.getResource(args[0]);
             collection.removeResource(document);

                watch.stop();
             System.out.println("Document " + args[0] + " removed");
        }
          catch (XMLDBException e)
          {
             System.err.println("XML:DB Exception occured " + e.errorCode + " " +
e.getMessage());
        }
          finally
          {
             if (collection != null)
                {
                 collection.close();
             }
        }

          System.out.println ("Time elapsed to remove resource : "+ watch.elapsed() + "
ms");
    }
}


/// add document.java
package org.apache.xindice.examples;

import java.io.File;
import java.io.FileInputStream;

import org.xmldb.api.base.Collection;
import org.xmldb.api.base.XMLDBException;
import org.xmldb.api.modules.XMLResource;

import org.apache.xindice.Stopwatch;

/**
 * Simple XML:DB API Example to insert a new document into the database.
 */
public class AddDocument  extends AbstractExample
{

    public static void main(String[] args) throws Exception
```

```
    {
        Collection collection = null;
          Stopwatch watch = new Stopwatch();
        try
          {

               if(args.length < 3)
               {
                      System.out.println(" Error: Usage [cmd] [file] [resourceid]
[collection]");
                      System.exit(0);
               }

               String colstr = "xmldb:xindice:///db/" + args[2];
            collection = getCollection(colstr);


               System.out.println(args[0]+" " + args[1] + args[2]);
            String data = readFileFromDisk(args[0]);

               watch.start();
            XMLResource document = (XMLResource) collection.createResource(args[1],
"XMLResource");
            document.setContent(data);
            collection.storeResource(document);

               watch.stop();
            System.out.println("Document " + args[0] + " inserted as " +
document.getId());
        }
        catch (XMLDBException e) {
            System.err.println("XML:DB Exception occured " + e.errorCode + " " +
e.getMessage());
        }
        finally {
            if (collection != null) {
                collection.close();
            }
        }

               System.out.println(" Time to load this resource : "+ watch.elapsed() +" ms
");
    }

    public static String readFileFromDisk(String fileName) throws Exception {
        File file = new File(fileName);
        FileInputStream insr = new FileInputStream(file);
        byte[] fileBuffer = new byte[(int) file.length()];
        insr.read(fileBuffer);
        insr.close();
        return new String(fileBuffer);
    }
}

////// Createcollection.java //////

package org.apache.xindice.examples;


import org.apache.xindice.client.xmldb.services.CollectionManager;
import org.apache.xindice.xml.dom.DOMParser;
import org.xmldb.api.base.Collection;
import org.xmldb.api.base.XMLDBException;

import org.apache.xindice.Stopwatch;

/**
 * Simple XML:DB API example to create a collection.
 */
public class CreateCollection  extends AbstractExample
 {
```

```
    //private static final String COLLECTION_NAME = "mycollection";


    public static void main(String args[]) throws Exception
         {
        Collection collection = null;
          Stopwatch watch = new Stopwatch();

               if (args.length < 1 )
               {
                       System.out.println(" Incorrect usage: [cmd] [collection] " );
                       System.exit(0);
               }
          String COLLECTION_NAME = args[0];
        try
          {


                       // collection = getCollection("xmldb:xindice:///db/");
                collection = getCollection("xmldb:xindice://localhost:8888/db/");
                       // collection = getCollection("xmldb:xindice-embed:///db/");

                       watch.start();
                    CollectionManager service = (CollectionManager)
collection.getService("CollectionManager", "1.0");

             try
                       {
                        service.dropCollection(COLLECTION_NAME);
                   System.out.println("Dropped existing collection with name: " +
COLLECTION_NAME);
                    }
             catch (Exception e)
                       {
                        ; // nothing, this may be the first pass.
             }

               // Build up the Collection XML configuration.
                        String collectionConfig =
                        "<collection compressed=\"true\" name=\"" + COLLECTION_NAME +
"\">"
                    + "   <filer class=\"org.apache.xindice.core.filer.BTreeFiler\"/>"
                    + "</collection>";

               service.createCollection(COLLECTION_NAME,
DOMParser.toDocument(collectionConfig));

                       watch.stop();
                       System.out.println("Collection " + COLLECTION_NAME + " created.");
        }
          catch (XMLDBException e)
          {
            System.err.println("XML:DB Exception occured " + e.errorCode + " " +
e.getMessage());
        }
          finally
          {
            if (collection != null)
               {
                collection.close();
            }
        }

          System.out.println(" Time taken to create collection "+ COLLECTION_NAME +" : " +
watch.elapsed() + " ms" );
    }
}

//////////////////////////////////////////
```

TECHNOLOGY 3 – Java Architecture for XML Binding
---

```
// To unmarshal XML to java content tree

import java.io.FileInputStream;
import java.io.IOException;
import java.util.Iterator;
import java.util.List;

import javax.xml.bind.JAXBContext;
import javax.xml.bind.JAXBException;
import javax.xml.bind.Unmarshaller;

// import java content classes generated by binding compiler
import primer.po.*;


public class Main {

    // This sample application demonstrates how to unmarshal an instance
    // document into a Java content tree and access data contained within it.

    public static void main( String[] args ) {
        try {
            // create a JAXBContext capable of handling classes generated into
            // the primer.po package
            JAXBContext jc = JAXBContext.newInstance( "primer.po" );

            // create an Unmarshaller
            Unmarshaller u = jc.createUnmarshaller();

            // unmarshal a po instance document into a tree of Java content
            // objects composed of classes from the primer.po package.
            PurchaseOrder po =
                (PurchaseOrder)u.unmarshal( new FileInputStream( "po.xml" ) );

            // examine some of the content in the PurchaseOrder
            System.out.println( "Ship the following items to: " );

            // display the shipping address
            USAddress address = po.getShipTo();
            displayAddress( address );

            // display the items
            Items items = po.getItems();
            displayItems( items );

        } catch( JAXBException je ) {
            je.printStackTrace();
        } catch( IOException ioe ) {
            ioe.printStackTrace();
        }
    }

    public static void displayAddress( USAddress address ) {
        // display the address
        System.out.println( "\t" + address.getName() );
        System.out.println( "\t" + address.getStreet() );
        System.out.println( "\t" + address.getCity() +
                            ", " + address.getState() +
                            " "  + address.getZip() );
        System.out.println( "\t" + address.getCountry() + "\n");
    }

    public static void displayItems( Items items ) {
        // the items object contains a List of primer.po.ItemType objects
        List itemTypeList = items.getItem();


        // iterate over List
```

```
        for( Iterator iter = itemTypeList.iterator(); iter.hasNext(); ) {
            Items.ItemType item = (Items.ItemType)iter.next();
            System.out.println( "\t" + item.getQuantity() +
                                " copies of \"" + item.getProductName() +
                                "\"" );
        }
    }
}

//// Modify marshal.java

import java.io.FileInputStream;
import java.io.IOException;
import java.math.BigDecimal;

import javax.xml.bind.JAXBContext;
import javax.xml.bind.JAXBException;
import javax.xml.bind.Marshaller;
import javax.xml.bind.Unmarshaller;

// import java content classes generated by binding compiler
import primer.po.*;


public class Main {

    // This sample application demonstrates how to modify a java content
    // tree and marshal it back to a xml data

    public static void main( String[] args ) {
        try {
            // create a JAXBContext capable of handling classes generated into
            // the primer.po package
            JAXBContext jc = JAXBContext.newInstance( "primer.po" );

            // create an Unmarshaller
            Unmarshaller u = jc.createUnmarshaller();

            // unmarshal a po instance document into a tree of Java content
            // objects composed of classes from the primer.po package.
            PurchaseOrder po =
                (PurchaseOrder)u.unmarshal( new FileInputStream( "po.xml" ) );

            // change the billto address
            USAddress address = po.getBillTo();
            address.setName( "John Bob" );
            address.setStreet( "242 Main Street" );
            address.setCity( "Beverly Hills" );
            address.setState( "CA" );
            address.setZip( new BigDecimal( "90210" ) );

            // create a Marshaller and marshal to a file
            Marshaller m = jc.createMarshaller();
            m.setProperty( Marshaller.JAXB_FORMATTED_OUTPUT, Boolean.TRUE );
            m.marshal( po, System.out );

        } catch( JAXBException je ) {
            je.printStackTrace();
        } catch( IOException ioe ) {
            ioe.printStackTrace();
        }
    }
}
```

VITA

Uma Chinmayee Nirmal

Candidate for the Degree of

Master of Science

Thesis: EFFICIENT STORAGE OF XML – A COMPARATIVE STUDY

Major Field: Computer Science

Biographical:

Personal Data: Born in Hyderabad, Andhra Pradesh, On October 6, 1979, the daughter of Sreekanth and Bhairavi Nirmal.

Education: Graduated from St. Ann's High School, Secunderabad, Andhra Pradesh, India, in March 1995; graduated from St. Francis' College for Women, Secunderabad, Andhra Pradesh, India, in April 1997; received Bachelor of Engineering degree in Civil Engineering from Osmania University, Hyderabad, Andhra Pradesh, India in May 2001. Completed the requirements for the Master of Science degree with a major in Computer Science at Oklahoma State University in May, 2005.

Experience: Employed by Oklahoma State University, Central Dining Services, as a student worker in 2002; employed as a Graduate Assistant, by the Oklahoma State University, School of Education, College of Education, as a student worker by the Oklahoma State University, Department of Veterinary Pathobiology, College of Veterinary Medicine and as a Graduate Lab Assistant by the Oklahoma State University, College of Engineering, Architecture and Technology Information Technology Services Department during the Spring, Summer and Fall semesters of 2003 respectively; employed as a Graduate Assistant by the Oklahoma State University, Department of Physics and Oklahoma State University, Computer Science Department, from 2004 to present.

Professional Memberships: None

Name: Uma Chinmayee Nirmal Date of Degree: May, 2005

Institution: Oklahoma State University Location: Stillwater, Oklahoma

Title of Study: EFFICIENT STORAGE OF XML - A COMPARATIVE STUDY

Pages in Study: 80 Candidate for the Degree of Master of Science

Major Field: Computer Science

Scope and Method of Study: The purpose of this study is to predict the performance of XML storage in various real time scenarios. This study is a survey and comparative analysis of data storage using databases to store and retrieve XML, using Java objects representing XML and other storage mechanisms that may have not yet been explored. It also gives a high-level overview of how to use XML with databases or Java Objects and describes how the differences between data-centric and document-centric XML affect their usage, when used with databases and objects, and how XML is used with relational and object oriented databases, Java Objects, and the role of native XML databases (stand alone XML databases).

Findings and Conclusions: A detailed comparative study on storage of XML using Relational DBMS, Native XML DBMS and processing into Java Objects using JAXB was conducted. The data models such as relational, hierarchical, document-driven were used as inputs to the study. There is no single tool that can manage all the aspects of XML data used in an application. Each technology provides interestingly unique features.

There is a tremendous amount of research and development in progress, in the development of tools and technologies to use XML. It can be safely predicted that all the technologies will finally merge into one standard method of storage of XML that will incorporate all the features such as, faster searches, full-text searches, maintaining original document order, ability to maintain a collection of documents, ability to query and store or retrieve over the network using protocols such as HTTP, SOAP etc., provide integral support for casting of elements, support for processing valid and non-valid XML documents, all in a single tool.

This study has successfully concluded that the most efficient way to store XML data lies in the context of its usage.

ADVISER'S APPROVAL: Dr. Blayne E. Mayfield