Temporal XML: Version Management and Query Processing

Project Members

Zimasa Ndamase
Computer Science Department
University of Cape Town
Private Bag
Rondebosch 7701, RSA
zndamase@cs.uct.ac.za

Matšeliso Thabane
Computer Science Department
University of Cape Town
Private Bag
Rondebosch 7701, RSA
mthabane@cs.uct.ac.za

Project Supervisor

Associate Professor Sonia Berman
Computer Science Department
University of Cape Town
Private Bag
Rondebosch 7701, RSA
sonia@cs.uct.ac.za

ABSTRACT

The changes that can be made to a document throughout its lifetime are limitless. Temporal management of documents with multiple versions is a complex area of research that has risen from the need to preserve information. Today's technology-driven society is rapidly moving towards web environments and using XML as the format of choice of information representation and exchange. The proposed solutions is an implementation of a temporal XML version management system that stores and reconstructs multiple versions of XML documents, as well as provide temporal query capabilities that show how the document has changed over time.

KEYWORDS

XML, temporal XML, temporal query, temporal databases, document retrieval, version management, multiple versions.

INTRODUCTION

As users migrate to web-based environments for information exchange and the use of XML format documents increases, the need to be able to store and manage XML content becomes apparent. XML documents are seldom static as old documents get deleted, new ones created and older versions updated. This report looks at the complexities of the implementation of a temporal XML version management system that has two modules, (a) a storage and reconstruction component for storing and retrieving versions of XML documents, and (b) a temporal query processing component. Finding an efficient storage and reconstruction mechanism that allows efficient temporal query processing was important. The biggest storage challenge was storing the elements such that each one had an element identifier and its position in the XML file. This facilitated the reconstruction of document versions. The temporal query processor performs temporal queries on multiple versions of documents to reveal the content of a document on specific dates and the history of the document and how it has evolved over time. The greatest query challenge was specifying temporal queries as a combination of SQL and XQuery statements. The SQL statement gets sent to the database which returns document versions that XQuery statements extract data

from. An additional area of concern was developing an intuitive user interface where users can specify temporal queries.

One of the greatest advantages of using XML is that XML documents have a structure that can be associated with a schema [1], thereby making mapping to relations for database storage simpler. Additionally, XML has the flexibility to represent very different kinds of data from different sources. This structured nature also enables the performance of more precise queries. A temporal middle-layer approach has been used where the temporal query processor, with a front-end user interface, is built on top of a database management system. This is referred to as using the stratum approach. The temporal query component translates temporal query user input into two portions, (a) conventional SQL statements that retrieve the relevant document version(s), and (b) one or more XQuery statements that extract the relevant data from one or more document versions.

There are countless areas of application for temporal version management systems for information preservation; these include data warehouses, digital archives, XML repositories, project management, inter/intra-company warehouses, software configuration management and the maintenance of personnel records. Content-seekers are users that may want to download, retrieve or keep track of documents and perform queries on the document's versions. Content-authors may want to maintain documents that are constantly changing with as much efficiency and flexibility as possible, as opposed to simply maintaining the latest version of the document. These and other areas of application would benefit from the use of a standardized solution for the management of temporal data.

THEORY AND BACKGROUND

The integrated approach [10] represents a long-term solution that involves building an underlying DBMS that has built-in temporal operators, temporal functions, and a temporal query language. An integrated approach supports the storage, retrieval and querying of time-varying data more efficiently and eliminates the disadvantages of the stratum approach. Therefore, programmer productivity and code maintainability improves as the bulk of the temporal processing gets moved from the application to the database end

One solution that is feasible in the short-term is often referred to as the stratum approach [8]. This constitutes a middle-layer query processor between the application interface and the DBMS. The middle-layer converts temporal query statements into conventional SQL statements, using a database management system that is not specialized to processing temporal information or storing temporal data. The shortcomings to this approach can be attributed to the processing overhead associated with evaluating a temporal query and representing it as SQL statement(s), this means that only a limited set of operations can be supported.

Document Types

A document can be characterized as being either document-centric or data-centric [1]; however, documents can be mixture or the two as well. Data-centric documents are not meant to be processed by machines and are often merely a collection of data values and are simply used to transport data. Examples of these are train schedules or share prices. Document-centric documents are meant to be human-readable and have a less regular structure than that of data-centric documents. Examples of these include books and journals.

OVERVIEW

DATABASE STORAGE

The storage alternatives that have been considered are; native XML databases, XML-enabled databases, or a new

implementation of a temporal DBMS. Native XML databases are young and not very widely used yet, therefore, there is limited support and documentation that is available. Developing and implementing a new DBMS that has temporal functionality is a far more complex task that requires a great deal of time and resources. This would serve as a suitable long-term solution. The XML-enabled databases map XML to a traditional database [11]. Oracle 9i is the XML-enabled database that has been used for document storage .

When a user wants to store an XML document version into the database, they specify the file to store and the system will send the file to a database (see figure 1). The system implements two storage approaches. The first approach uses built-in data types of Oracle which support XML manipulation. The second approach represents the document as a tree structure and stores each node together with timestamp information as a tuple.

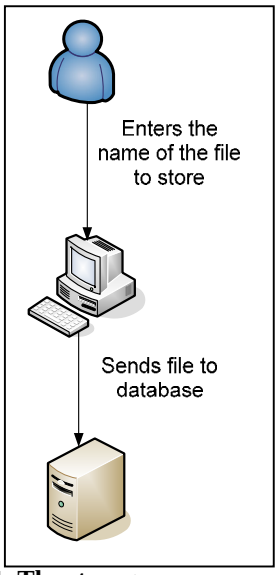

Figure 1. The storage process.

Reconstruction

The user wants to retrieve an XML document from the database. There are three types of files that can be reconstructed.

(i) Version files: a version of the document that is current at a particular point in time, these are called V_i .[8]

(ii) Edit files: these files show the content and position of changes that have been made using <OLD> and <NEW> tags, these are called E_i (se figure 2).

(iii) History files: a single XML file that encodes within it the full history of all changes made to that document up to some point in time - these are called H_i (see figure 3).

Version files are normal XML files that show content that is current at a certain point in time. Edit files are version files that show the content and position of changes to a version using <OLD> and <NEW> tags. History files are XML files that have author, start and end date attributes for every element in the file. A history file shows the evolution of each element in a document.

```
<DOCUMENT>
 <Intro>
    <OLD><name>Stan
            </name>
    </OLD>
    <NEW><name>Eric
            </name>
    </NEW>
</Intro>
</DOCUMENT>
```

Figure 2. Edit file - E_i

```
<DOCUMENT from='2001-01-01' to='now'>
 <Intro from='2001-01-01' to='now'>
     <name from='2001-01-01' to='2001-04-11'>Stan
     </name>
     <name from='2001-04-11' to='now'>Eric
     </name>
</Intro>
<Body from='2001-01-01' to='now'>
 </Body>
</DOCUMENT>
```

Figure 3. History file - H_i

QUERY PROCESSOR

The initial implementation of the query processor involved making a number of design decisions. The first consideration was deciding on whether to use or extend XPath or XQuery. These are query languages that query from XML documents. The second consideration was whether to develop a new temporal query language instead of using existing query languages. The decision to use XQuery for XML document querying was based on the following reasoning:

- Although XPath is a powerful W3C specification used to query from XML documents, it does not allow typical query constraints such as 'WHERE' and 'ORDER-BY' operators.
- The algorithms used for the evaluation of temporal queries made it possible to use XQuery statements to extract the nodes of an XML document; therefore it was not necessary to design a new temporal query language or to extend XPath or XQuery.

In order to use XQuery, an implementation of the XQuery API was required. The *XQEngine* [2] is the XQuery API that has been used to query from XML documents. The chosen XQuery API implementation is open-source, well-documented and has been tested on and passed the prescribed W3C XQuery Test Suite (XQTS) [3]. Some of the alternatives to *XQEngine* were provided for trial-period use as evaluation versions that would have expired before the end of this project. Other alternatives were not open-source.

TEMPORAL QUERIES

The user enters temporal query parameters on the graphical user interface of the *Temporal Query Application*. The GUI constitutes the front-end of the query processor. It has been used for the purpose of abstracting the query language details from users. This removes the requirement for users to have query language knowledge, making it possible for more people to easily learn how to use the system.

The query processor requests version files or history files from the database depending on the type of query being performed. History files are used for *author* and *change* queries, and version files are used for *snapshot* and *history* queries.

User-centered design techniques were used to develop the GUI and test it for usability. To obtain user experiences and feedback, the following techniques were used:
An initial discussion where participants were shown preliminary GUI designs, the feedback obtained was used to evolve the GUI prototype.
The evolved prototype was later tested on participants using a questionnaire and evaluation sheet.
This testing session was followed by a focus-group style discussion of general feedback and other comments.

Using a GUI provides limited functionality as only the specified types of queries can be performed. This is sufficient for users that do not require extensive query functionality; this also provides an easy to use solution for the average user who may not have query language knowledge.

RESULTS

These are the results of using an XML-enabled database:

(i) Using the XML Type to store the original document and subsequent changes as CLOBs does not maintain the structure of a document when it gets reconstructed.

(ii) The greater number of changes in an edit file, the longer the database takes to store document versions.

The 'Temporal Query Application' is a query processing tool that can perform a number of simple and complex temporal queries. The biggest challenge was deciding

which types of queries to support. The GUI approach that has been taken limits the kinds of queries that can be specified. This means that users can only perform a finite number of queries as opposed to the possibilities that one has when using a query language.

The testing and evaluation participants were generally satisfied with the GUI. 66.67% of the GUI evaluation participants indicated that the interface encompasses the following usability properties: learnability, memorability, efficiency and effectiveness.

CONCLUSION AND FUTURE WORK

Temporal querying and version management are complex research areas that pose a range of problems. One of these problems is that there are many ways that time can be captured and represented; selecting the most fitting way to do this may vary for different applications. A greater concern is that temporal queries are essentially translating queries that are best expressed as sentences into temporal query languages.

An effective and flexible approach that is general enough to be used in any context requires a temporal DBMS. This DBMS would be optimized to store, retrieve and query from temporal data.

The use of a GUI limits the types of queries that can be performed; however, the queries that have been implemented provide non-XML accustomed users and XML proficient users with a way to query multi-versioned XML documents in a meaningful way.

REFERENCES

[1] Nørvåg K. *Temporal Query Operators in XML Databases.* Department of Computer Science, Norwegian University of Science and Technology, Norway. Proceedings of 2002 ACM symposium on Applied computing.

[2] XQEngine. XQuery API implementation. Available: http://www.fatdog.com/. Date Accessed : 9 October, 2006.

[3] XML Query Test Suite. World Wide Web Consortium. Available: http://www.w3.org/XML/Query/test-suite/. Date Accessed: 15 October, 2006.

[4] S.-Y. Chien, V. J. Tsotras, C. Zaniolo, "*Efficient schemes for managing multiversion XML documents"*, The VLDB Journal — The International Journal on Very Large Data Bases, Volume 11 Issue 4, pg 332 – 353, December 2002.

[5] Raymond K. Wong, Nicole Lam, "*Managing and Querying Multi-Version XML Data with Update Logging*", Proceedings of the 2002 ACM symposium on Document engineering
Publisher: ACM Press, New York ,USA, pg 74-81 ,November 2002.

[6] Shu Yao Chien, Vassilis J. Tsotras, Carlo Zaniolo,"*Version Management of XML Document*", ACM SIGMOD Record, Volume 30 Issue 3 , pg 184-200, September 2001.

[7] Shu Yao Chien, Vassilis J. Tsotras, Carlo Zaniolo ,"*A comparative Study of Version Management Schemes for XML Documents"*, A TimeCenter Technical Report, *University of California, 5* September 2000.

[8] Chien S. et al, "*Storing and Querying Multiversion XML Documents using Durable Node Numbers"*, Proc. of the second International Conference on Web Information Systems Engineering (WISE I),Washngton DC,USA,Volume 1,pg 232, 2001 .

[9] Al-Ekram R., Adma A., Baysal O, "*diffX: An Algorithm to detect Changes in Multi Verios XML Documents"*, Proceedings of the 2005 conference of the Centre for Advanced Studies on Collaborative research CASCON '05, Toranto Ontario, Canada, pg 1-11, October 2005.

[10] Slivinskas G. et al. *Adabptable Query Optimization and Evaluation in Temporal Middleware.* Department of Computer Science, Alborg University, Denmark. Department of Computer Science, University of Arizona, USA.

[11] Nørvåg K., Limstrand M. and Myklebust L. *TeXOR: Temporal XML Database on an Object-Relational Database System.* Lecture Notes in Computer Science, Perspectives of System Informatics, Volume 2890/2003, 2003. Department of Computer Science, Norwegian University of Science and Technology, Norway