AN EFFECTIVE APPROACH FOR MODIFYING XML DOCUMENTS IN THE CONTEXT OF MESSAGE BROKERING

R. Gururaj, Indian Institute of Technology Madras, gururaj@cs.iitm.ernet.in
M. Giridhar Reddy, Indian Institute of Technology Madras, giridhar@cse.iitm.ernet.in
P. Sreenivasa Kumar, Indian Institute of Technology Madras, psk@cs.iitm.ernet.in

ABSTRACT

XML message brokering systems support customization that modifies the structure and content of XML messages under dissemination. This allows customized content delivery and data and application integration. In this work we present an approach for modifying XML documents in the context of XML message brokering. In our approach we propose to store XML data in its native form in a BDB XML database and perform updates specified in XQuery language. We compare our approach with another technique that stores XML data in relational tables.

Keywords: XML, XQuery, Information Dissemination, Message Brokering

INTRODUCTION

Information dissemination involves distributing data produced by data sources to a set of interested data consumers in a distributed environment. A *message broker* plays a key role in information dissemination as a message exchange point for messages sent between producers and consumers. Considering the significance of XML [12] as a standard for representation and exchange of data on the web, we assume that the future dissemination systems support XML data exchange. A message broker, that handles XML data, is called an *XML message broker*. An XML message broker can modify the structure and content of the received message before it is delivered to the client. This customization is according to the user preferences. This allows data and application integration, personalized content delivery, and cooperation among disparate web services. Recently, we proposed a system VAXBro [5], a value-adding XML message broker that addresses the data processing needs of advanced customization (value-addition) process that modifies the structure and content of an XML message in an XML message broker. In VAXBro, users specify their customization requirements using subset of XQuery format as discussed in [4, 5]. The above-mentioned customization needs extensive update operations on XML message. The strategies employed to store XML data play a significant role in performing update operations in the context of message brokering.

In this work we describe the approach we have proposed in VAXBro, for storing and updating XML data. Then we compare our approach with another existing approach [11] that stores XML data in relational tables in RDBMS. We also discuss the advantages and suitability of our approach in the context of XML message brokering scenario.

The rest of the paper is organized as follows: in Section 2 we discuss the background and related work. Section 3 provides details of our approach. In Section 4 we present the comparison and performance results, and finally, Section 5 concludes the paper.

BACKGROUND AND RELATED WORK

The present XML message brokers such as XFilter [1], WebFilter [7], and YFilter [2] accept *longstanding data interests* (profiles/queries) from users, in the form of XPath [12] expressions. On arrival of a message, they perform filtering to find the set of user queries that match the incoming message. One recent XML message broker [3] delivers the result of a user query (XQuery) to the user with customized tags. The need for advanced customization (value-addition) and inadequacies of the present XML brokering systems stimulated us to propose the system- VAXBro [6], a value-adding XML message broker. The customization actions proposed in VAXBro include (i) inserting new elements into documents, (ii) deleting elements, and (iii) renaming tags.

XML Update Query Language

As the XML message brokers handle information encoded in XML format, user profiles or queries are needed to be in some XML query format. The XPath is a World Wide Web Consortium (W3C) standard specification for addressing parts of the XML document. XPath provides a flexible way to specify path expressions. The XPath expression "/publisher/book//price" addresses all *price* elements

that are descendants of all *book* elements which are directly children of *publisher* element in a document with book-publisher information.

XQuery [12] is expected to be the future W3C standard for querying XML data. XQuery internally uses path expressions in query formulation. A sample XQuery query is given in Figure 1(a). The above query lists titles of books published by PHI in 1988, from the XML document 'bib.xml'. In this query, *$b* is a variable to which *book* element is bound. The RETURN clause specifies the output format. But, the present XQuery doesn't support update constructs, which are required to specify customization needs in a message broker. One recent work [11] proposed extensions to XQuery to specify update operations on XML data, using the syntax shown in Figure 1(b). In our work on VAXBro, we proposed a customization language that extends subset of XQuery [12] and its update syntax [11], and fits in our message brokering model. A sample customization request, which is according to our proposed format, is shown in Figure 1(c).

```
FOR $binding1 IN XPath-expr,…
LET $binding := XPath-expr,…
WHERE predicate1,….
UpdateOp,….

        EBNF for UpdateOp:
UPDATE $binding { subOp {, subOp}*}

and subOp is:
DELETE $child | RENAME $child TO name |
INSERT content [BEFORE | AFTER $child] |
REPLACE $child WITH $content |
FOR $binding  IN XPath-expr,…
        WHERE predicate1,.. UpdateOP
```

```
For $b IN document("bib.xml")//book
WHERE $b/publisher='PHI'
AND $b/year=1988
RETURN  <title>   $b/title   </title>
```

(a) A Sample XQuery query

```
FOR        $st        IN
document('stockQuotes.xml')/stockQuotes/stock,
$pr        IN
document('companyProfiles.xml')/companyProfiles/
        company[symbol=$st/symbol]
UPDATE $st{
        INSERT $pr/ceo
    RENAME $st/symbol TO 'org'

        DELETE $st/price
}
RETURN  <result>{$st/ceo}</result>
```

```
< stockQuotes >
  < stock >
     <symbol>  </>
     < date>   < />
     <time>    </>
     < price > < />
  < /stock>
   . . .
< /stockQuotes >
```

(d) stockQuotes.xml

```
< companyProfiles >
  < company >
     < symbol >  </ >
     <name>      </ >
     < hq >      </ >
     <address>   </ >
     <ceo>       </ >
  < /company>
   . . .
< /companyProfiles >
```

(e) companyProfiles.xml

Figure. 1. Xquery Update Syntax, Sample Queries and XML Documents

Storing XML Data

XML storage approaches play a significant role in making query and update operations effective. At present there exist two popular approaches as given below.

Storing XML in relational database systems: The hierarchical data represented in XML format is shredded into multiple relational tables. The queries that are formulated in XQuery language are translated into equivalent SQL queries on relational database. A middle layer does this transformation. XML data from one document is now stored in multiple tables. Querying data from the repository requires extensive join operations on these tables. At the same time, reconstructing the original document or producing the result of a query in XML format requires additional effort. Several methods have been proposed for mapping XML data into relations [9], and publishing relational data as XML documents [10].

Storing XML in native form: A Native XML database defines a (logical) model for an XML document as apposed to the data in the document, and stores and retrieves documents according to that model. The database is specialized for storing XML data and

stores all components of the XML model intact. Documents go in and documents come out. Several vendors have already begun to release prototype XQuery implementations for use with their databases. BDB XML [8] database system supports basic XQuery operations and provides an API to perform modifications to XML document independent of XQuery.

The objectives of this work are as follows: (i) implement the XML storage and update strategy that uses BDB XML native XML database and its API, (ii) compare the effectiveness of our approach with XML update techniques described in [11] which stores XML data in relational tables, and (iii) finally, establish the suitability of our approach in the context of XML message brokering.

OUR APPROACH TO STORING AND UPDATING XML DATA

Our proposed value-adding XML message broker **VAXBro** [5], accepts user customization requests in a format that uses subset of XQuery format. On receiving a message from a source, the system executes all the involved user queries. We call the incoming document on which customizations are specified as *target document*. System output consists of a set of customized XML documents, which will be delivered to concerned users. The consumer of the customized message could be an end-user (human or an application that accepts XML data input from external sources), or another XML message broker on the downstream of information dissemination network.

In this section, we describe our approach that addresses the following issues in the context of XML message brokering. First we discuss the XML message storage methodology in our proposed system VAXBro. Then we explain how customization requests submitted using subset of extended XQuery format are handled to modify the structure and content of the document.

BDB XML Database to Store XML Data

In our proposed system VAXBro, we store XML messages in BDB XML [8]. BDB XML is an embedded database to manage and query XML documents. BDB XML can be used through programming API. As BDB XML is an embedded engine, we use it with our application in the same way as we would use any other third-party package. In BDB XML documents are stored in *container*, which we create and manage through *XmlManager* objects. Each such object can open multiple containers at a time. Each container can hold a large number of XML documents. Once the document is placed into a container, we can use XQuery to retrieve documents or required parts of documents. Queries are performed through *XmlManager* objects. BDB XML supports XQuery working draft. As XQuery is an extension to XPath2.0, BDB XML provides full support for that query language also.

As the present XQuery draft doesn't include update features, BDB XML too doesn't support modifications to XML through XQuery. But, BDB XML provides document modification facility through its API. This allows us to easily add, delete, or modify selected portions of XML document. Moreover, BDB XML stores XML document in native form. BDB XML also supports lot of other features like- indexing, transaction management etc. The official BDB XML provides the library in C++, Java, Perl, Python, PHP and Tcl languages.

Updating XML Date Stored in BDB XML Using our Extended XQuery Requests

Now, we describe our approach to updating XML data stored in BDB XML. First we discuss the BDB XML's API support for performing following modification operations.

(1) *INSERT operation*: this is to add an element to the document at specified location. We use the method *addAppendStep()*. The parameters need to be passed are: (i) location where the said content is to be inserted, (ii) content to be inserted, and (iii) the tag name of the inserted content.

(2) *RENAME operation*: this operation changes the name of a tag of an element specified by the path. For this we use the method - *addRenameStep()*. This method requires the element path to be renamed and the new name.

(3) *DELETE operation*: will remove all elements addressed by the path. This is executed by *addRemoveStep()* method. This method requires the element to be deleted.

All the above methods are defined in **XmlModify** class of BDB XML API. To understand this, let us consider the customization request shown in Figure 1 (c). The structure of all the involved documents are shown in Figure 1 (d, e). The query has one *INSERT* operation, where the element *ceo* from *companyProfile.xml* is inserted into *stockQuotes.xml*. Here, the target location where insertion takes place

is given by *$st*, which is bound to path-*document('stockQuotes.xml')/stockQuotes/stock*. Then the content to be inserted is specified by *$pr/ceo* where *$pr* is bound to */companyProfiles/company* path in *companyProfiles.xml*, and the name of the tag is *ceo* by default, because it is not explicitly mentioned by the user. The second operation of the query is RENAME. The element to be renamed is specified by the path *$st/symbol*, and the new name is *org*. The last operation of the query is DELETE. In our example query, the element to be removed is given by the path *$st/price*. It is clear that all the required parameters to call the methods *addAppendStep(), addRenameStep() and addRemoveStep()* are directly or indirectly available in the specified query.

Once, we add modification steps by calling appropriate methods, finally we call the method *execute()* of *XmlModify* object by passing on the document to be modified and other required parameters.

In our approach, first we parse the user submitted customization query and extract all the required information to call BDB XML API methods, and store the same in suitable data structures. As all the queries in message brokers are longstanding queries, this parsing activity is done only once for each query. Next, whenever it is needed to customize the documents in BDB XML database, our *query engine* calls appropriate methods of API, by passing on required information, which stored in local data structures. The entire update handling process is depicted in Figure 2.

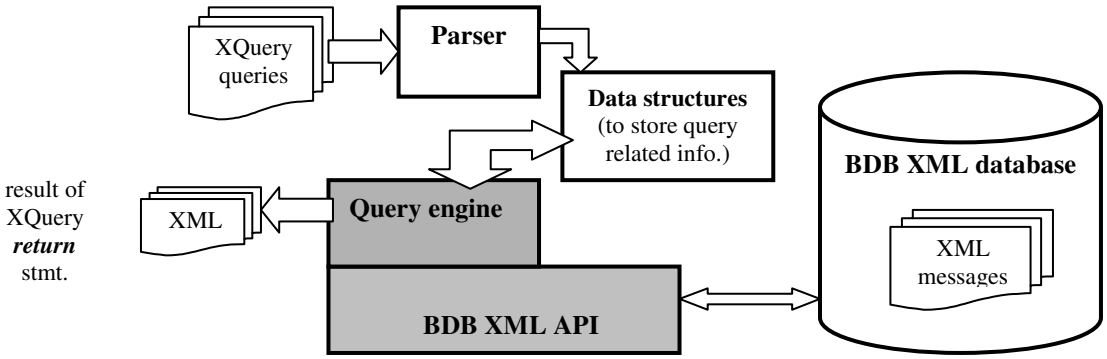

Figure. 2. Handling XML updates.

We have implemented XQuery *Parser* and *Query engine* components, as shown in Figure 2, to validate our ideas. The Parser is a Java program that calls JavaCC [6] parser program. We store parsed query related information in IBM DB2 relational database. The Query engine is also a Java program that makes calls to BDB XML API to perform modifications. We have also implemented the *return* functionality to extract the required portion of the document and return it as an XML document.

PERFORMANCE EVALUATION

We have implemented the *relational approach* to update XML described in [11]. In this approach XML data is stored in relational tables. As the objective of our work is to assess the effectiveness of the update approach, we implement the following basic functionality- (i) inserting an arbitrary element at specified location, and (ii) deleting an element at specified place in the document. We also implemented the functionality of returning specified leaf elements of the modified document. We implemented the *relational approach* in Java and we have used JavaCC to parse queries, and Oracle9i to store XML data. The query execution time for a standard query on similar tables for DB2 and Oracle9i is almost same. Hence, this disparity in choosing the RDBMS in two implementations doesn't make significant difference.

We formulated a set of XQuery update requests on document with structure shown in Figure 1 (d). The update requests are shown in Figure 3. The document captures stock quotes from YAHOO site. The number of stock elements was 50. With an intension to evaluate the effectiveness of our approach and *relational approach*, we executed the same set of queries using both implementations. We conducted the above experiments on Windows 2000 machine

with Intel Pentium-IV 1.7Ghz processor and 256 MB RAM. The results are shown as a graph in Figure 4.

```
Q1: FOR $st IN
document("stockQuotes.xml")/stockQuotes/stock
UPDATE $st
{
INSERT <status>ok</status>
}
RETURN <result>{$st/status}</result>

Q2: FOR $st IN
document("stockQuotes.xml")/stockQuotes
                /stock[symbol="ADTN"]
UPDATE $st
{
INSERT <hod>abc</hod>
}
RETURN <result>{$st/hod}</result>

Q3: FOR $st IN
document("stockQuotes.xml")/stockQuotes/stock
UPDATE $st
{
INSERT <sellingPrice>high</sellingPrice>
DELETE $st/hod
}
RETURN <result>{$st/sellingprice}</result>

Q4: FOR $st IN
document("stockQuotes.xml")/stockQuotes/stock
UPDATE $st
{
INSERT <cat>Regular</cat>
INSERT <place>usa</place>
}
RETURN <result>{$st/place}</result>
```

Figure 3. Test Queries

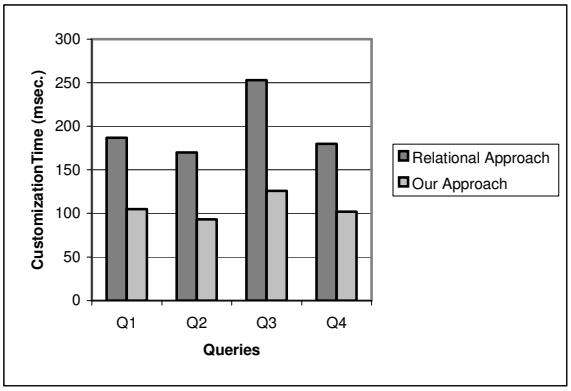

Figure 4. Query Execution Time

From the results it is evident that our approach to updating and rebuilding XML documents works better than that of *relational approach*. In the present relational implementation we have not implemented rename operation, inserting complex elements, and inserting elements extracted from other documents. It is obvious that the complexity of such operations would further reduce the performance in case of relational approach. This is because, new tables are to created for new elements, which are not in old structure. Further, in message brokering systems, after modifications, the document need to be built back for dissemination purpose. It is clear that this process involves pooling up data spread across multiple tables in relational approach. In our approach this needs less effort because we store documents in native form. Another disadvantage with relational approach is that, every time a new message arrives, we need to shred that data in to respective tables before doing any processing. This is not needed in our case. And, if multiple queries request different customization, it is too complex to handle that kind of situation in relational approach, as we need copies of same set of tables for each query. In our approach we just copy the whole target document for each query.

Hence, we argue that our approach to storing and modifying XML data is more effective than relational approach, for XML message brokering systems.

CONCLUSIONS

In this paper we presented details of our proposed approach to store and modify XML documents. We store XML data in native form in BDB XML. We have also implemented the basic functionality of *relational approach*. We compared the effectiveness of our approach with relational approach, in the context of XML message brokering. Experimental results showed that our approach performs better than relational approach. We plan to extend our implementation to support more update features. We argued in favor of the suitability of our approach for use in value-adding XML message brokering context.

REFRENCES

1. Altnel M. and Franklin M.J. (2000). Efficient Filtering of XML Documents for Selective Dissemination of Information. *Proceedings of the 26th VLDB Conference*, Cairo, Egypt, 53-64.
2. Diao Y., Franklin M.J. (2003). High-Performance XML Filtering: An Overview of YFilter. I*EEE Data Engineering Bulletin*, *26*(1), March, 41-48.

3. Diao Y., Franklin M.J. (2003). Query Processing for High-Performance XML Message Brokering. *Proceedings of the 29th VLDB Conference*, Berlin, Germany, 261-272.
4. Gururaj R. and Sreenivasa Kumar P. (2005). Service Specification in a Value Adding Broker for Data Dissemination Through XML. *Proceedings of the IACIS Pacific 2005 Conference*, Taipei, Taiwan, 406-413.
5. Gururaj R. and Sreenivasa Kumar P. (2005). VAXBro: A Value-Adding XML Message Broker. To appear in *Proceedings of the ADCOM 2005 Conference*, Coimbatore, India.
6. Java Compiler Compiler (JavaCC). Available: http://javacc.dev.java.net.
7. Pereira J., Fabret F., Jacobsen H.A., Llirbat F. and Shasha D. (2001). WebFilter: A High-throughput XML-based Publish and Subscribe System. *Proceedings of the 27th VLDB Conference*, Rome, Italy, 723-724.
8. Sleepycat software's Berkeley DB XML database. Available: http://www.sleepycat.com/products/xml.shtml.
9. Shanmugasundaram, J., Kristin Tufte, Chun Zhang, Gang He, David J. DeWitt, and Jeffrey F. Naughton. (1999). Relational database for querying XML documents: Limitations and opportunities. *Proceedings of the 25th VLDB Conference*, Edinburgh, Scotland, 302-304.
10. Shanmugasundaram, J., Eugene J. Shekita, Rimon Barr, Michael J. Carey, Bruce G. Lindsay, Hamid Pirahesh, and Berthold Reinwald. (2000). Efficiently publishing relational data as XML documents. *Proceedings of the 26th VLDB Conference*, Cairo, Egypt, 65-76.
11. Tatarinov, I., Zachary G. Ives, Alon Y. Halevy, and Daniel S. Weld. (2001). Updating XML. *Proceedings of the ACM SIGMOD Conference*, Santa Barbara, CA, USA, 413-424.
12. World Wide Web Consortium. Available: http:// www.w3c.org.