Malaysian Journal of Computer Science, Vol. 18 No. 2, December 2005, pp. 1-10

INTEGRATING XML WITH RELATIONAL DATABASES USING MIDDLEWARE APPROACH

Lee Ching Kum and Sai Peck Lee
Faculty of Computer Science & Information Technology
University of Malaya
50603 Kuala Lumpur
Malaysia
email: saipeck@um.edu.my

ABSTRACT

Over the past few years, XML has become the undisputable lingua franca standard both for semi-structured data representation and exchange format over the Internet, and also content management in various e-business worlds, especially the B2B and B2C enterprise applications. However, most of these organisations still rely heavily on existing relational database management systems (RDBMS) to store and manage their structured data for daily critical business transactions. In fact, major database vendors, which also happen to be the giant software companies like Microsoft, IBM and Oracle, have ventured and taken great initiatives in researching and providing for a single solution to integrate these semi-structured XML data with structured data in relational databases. Most importantly, it is estimated that during the next few years to come, more than 75% of e-business applications will implement XML technologies in their applications. Consequently, as more software applications are rapidly beginning to implement XML, there should be a growing need for XML middleware to efficiently integrating XML data at the front-end with a RDBMS at the back-end. Hence, this research is aimed at providing a generic XML-based framework, which is known as JXDB, that allows a user to use XML for dealing with semi-structured data for creating, accessing or updating to existing heterogeneous relational databases that store structured data and vice versa. JXDB is designed to provide a generic and extensive XML middleware framework for integration between XML documents and heterogeneous relational databases.

Keywords: ***XML, Relational Database Systems, XML Middleware, SQL, XQuery***

1.0 INTRODUCTION

XML is rapidly emerging as a widespread recognised standard over the past several years since it was first introduced as the replacement markup language for the popular but limited features of HTML. However, only over these few years that XML experts have begun to realise that XML was versatile and has more useful benefits outside its usage in the browsers, and was extremely suitable as a data exchange and data representation format among various applications on the Internet. Thus, due to the widespread acceptance of XML, a large growing number in XML documents and subsequently higher demands for efficient storage and retrieval of XML solutions are needed. Most importantly, the popularity of XML is due to its flexibility for representing many kinds of diverse information. XML is known as a versatile, readable meta-language for both humans and computers that can be understood easily because it is capable of self-describing the information content from various data sources, including semi-structured and structured documents, relational databases, native XML databases and object databases. Apart from that, the extensible nature of XML, coupled with its nested structure ability makes it possible to define new kinds of complex documents for specialised purposes, making it ideal for both representing and exchanging data over the Internet without the loss of semantics. Therefore, because of this, it is possible for applications to exchange and interpret this information. As the importance of XML has increased, e-businesses have much to gain from XML by automating and exchanging transactions with suppliers, partners and customers, such as in application-to-application (A2A), business-to-business (B2B) and business-to-customers (B2C) applications. As the Internet brings efficient connectivity for these applications to be integrated together, it has encouraged cost-effective e-commerce transactions, bringing it within the reach of all of the organisations in connected supply chain businesses. As a result, this causes major changes in the ways traditional businesses are conducted. Ultimately, most of the organisations still rely on RDBMS as their back-end database systems but now they need to integrate their existing RDBMS with these XML data. Thus, some efficient tools or middleware to view, query, export and import data between XML and relational databases are needed. These tools for querying and integrating between XML and relational databases are still largely in their infancy, but are beginning to emerge rapidly. Meanwhile, most of the major software companies and database vendors have been hard at work in addressing and finding the challenges of integrating XML with relational databases.

The scope of this paper is to study and analyse the possibilities and advantages of the enabling XML technologies in order to implement a working prototype of graphical user interface (GUI) XML-based middleware for the XML users to work with any heterogeneous relational database environment. JXDB system will not be replacing any of existing XML middleware or XML-enabled databases in the market, but serve as a basis of milestone to be used as a generic XML-based middleware that implements consistent and generic XML middleware framework approach for integration between XML documents and heterogeneous relational databases. Subsequently, efficient transfer and retrieval of XML data rely heavily on how the data is being stored and in what form do the applications need to use the data. The success of this prototype system is expected to increase the power of integration and development productivity among the software vendors who have to work in this area.

2.0 LITERATURE REVIEW

In this section, a brief literature review is done on the XML documents and relational databases. However, the review on the newly popular native XML databases and XML-enabled databases are beyond the scope of this paper. Here, the review covers the data transfer and mapping technologies used in integrating (marshalling and un-marshalling) XML documents with heterogeneous relational databases. In this case, when the user retrieves a set of records from the database, the system is able to retain the XML structures and able to generate respective XML documents and their DTDs at the end of the process. Before we proceed, we need to understand that naturally XML data are very different from relational data in several important aspects. This has influenced the design of the integration between XML and relational databases in order to have a smooth bi-directional data transfer between them. To start off, XML data are often heterogeneous and distribute their meta-data throughout the document itself as they are self-describing content. In contrast, the latter consists of a regular data structure, which allows the descriptive meta-data for the data to be stored in a separate catalog. XML documents can contain many levels of nested structures or elements, whereas relational data are generally flat in nature. Moreover, XML documents have an intrinsic order, whereas relational data are likely to be unordered except where an ordering can be derived from data values. Relational data are usually “dense”, meaning that nearly each and every column in relational tables has a value. In addition, relational tables are able to represent missing information by a special null value. On the other hand, XML data are often “sparse” and can represent missing information simply by the absence of an element. As a result, for these and also other reasons, we cannot directly transfer and integrate the XML data into existing relational databases, but it has to be coexisting with RDBMS to complement each other. Hence, we need to look into different and efficient methods or approaches to efficiently handle and integrate the XML data at the front-end to RDBMS at the back-end.

2.1 XML Middleware Approach

In this section, we will discuss briefly about XML middleware. Generally, XML middleware is an independent server applications or sets of APIs that sit between the front-end user application and the back-end databases to transfer XML data between the XML application and the back-end databases, which are mostly relational databases. One of its benefits is that it is able to provide services to a large scale of applications that is connected over the TCP/IP network. Subsequently, it reduces the complexity of application integration in a multi-tier and enterprise solution. This is because XML middleware approach permits applications to be divided into multiple layers thus allowing abstraction and reducing application complexity at each layer. Thus, multiple applications and users can connect and share one middleware over the network. XML middleware approach can be categorised into template-based and model-based approach [9]. The most common strategy that returns XML data from relational databases is template-based approach. This middleware approach processes an XML document with embedded SQL SELECT statements in a template, which are replaced in the output document by their results via the middleware software [9]. For example, it can generate an XML document from the data retrieved from a Customer table. On the other hand, template-based approach does not support transfer of data from XML documents to relational databases as it is too complex to do so. Most of the middleware transfer XML data directly according to the model on which they are built because both of them are structured differently in nature. Model-based approach creates a data model based on the XML document and then maps that model to the database. They are much more powerful as they can be used to transfer data from XML documents to databases and vice versa, and also as automated mapping tools. Middleware software is used by data-centric applications to transfer XML data between XML documents and databases, which sits between an XML document and a database [9]. These middleware use database drivers, such as ODBC, JDBC, or OLE DB, to connect to relational databases. A generic graphical view of the architecture of a middleware is shown in Fig. 1.

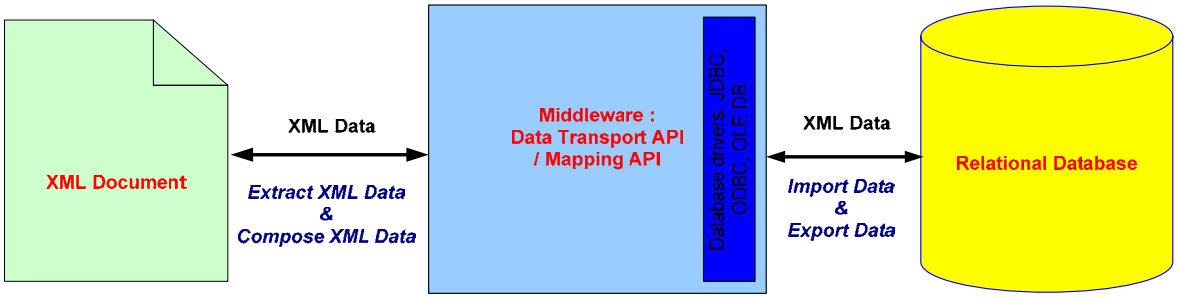

Fig. 1: Generic graphical view of middleware architecture

2.2 Review of XML Middleware: XML-DBMS

We reviewed one of the existing XML middleware in the open source community; called XML-DBMS. It is an open source middleware that consists of rich-feature of data transfer APIs to transfer data between XML documents and relational databases. It is designed using model-based approach that implements object-relational mapping technologies and is defined via a generated XML-based mapping language [9, 10]. Here, the mappings can be written manually from scratch or generated automatically from a given DTD or database schema [10]. Besides, both DTDs and database schemas can be generated from the mapping process, the latter approach can be reversible whereby DTDs can also be generated from database schema and vice versa. Like most middleware, XML-DBMS preserves the hierarchical structure of an XML document, as well as data in that document. If requested, it is also able to preserve the order in which children at a given level in the hierarchy appear. Additionally, when transferring data from an XML document to the database, users can specify what required actions to take like to insert, update data or both. On the other hand, when transferring data from the database to an XML document, users can use filters like the "where" expression to specify only the specific data to be retrieved [9, 10]. Thus it provides parameterised approach in this case for greater flexibility to the software developers. In spite of its many useful features, it is still just a set of Java API packages for transferring data between XML documents and relational databases used in an application that can be run from the command line or a program to transfer data between an XML document and a database.

3.0 JXDB SYSTEM ARCHITECTURE

JXDB is a GUI-based XML middleware designed in this proposed research to provide efficient transfer and retrieval of XML data from multiple XML documents at the front-end with heterogeneous relational databases at the back-end and vice versa. As a result, JXDB provides a consistent and extensible XML middleware framework. Fig. 2 shows the high-level 3-tier architectural design of JXDB system; it consists of several components that are integrated together to demonstrate the functionality of transferring XML data from multiple XML documents to heterogeneous relational databases, and vice versa. Each of these components is isolated according to their functionalities in the system. The presentation layer is responsible for displaying the application user interface whereby users will interact with the system through this interface. The business layer focuses on the components that are responsible to process and perform the business rules of the system. The data access layer is responsible for the data storage and retrieval management. This layer consists of components that know how to communicate with the place where the data storage resides. Different database products from different relational database vendors were tested; they are Oracle 9i, Microsoft SQL 2000, MySQL and Access 2000.

Users interact with the system through the XML-based Interfaces package at the presentation layer. This package consists of a set of use cases that will illustrate what kind of services the system can provide to these users, such as load, insert, delete or update XML data to relational tables and retrieve XML data from relational tables to XML documents. In the business layer, there are several components that are integrated together to perform necessary jobs, such as the query interpreter is responsible to interpret queries actions like insert, delete and update via data grid control that stores the persistent XML data in memory, and also to query and retrieve XML data via W3C XQuery on multiple XML documents. Other components include the mapping component that is used to map XML schemas or DTDs to database schemas using table-based mapping definitions. In order to transfer data from XML documents to any type of database, it is necessary to map the XML document schemas, which consist of respective DTDs or XML schemas to the database schemas. The transformation & transfer component is used to transform the decomposed XML data in DOM memory to Java objects based on the mapping schemas defined. Following that,

these persistent objects will transfer the updated data to relational tables. Besides, it also provides data export from relational tables to compose XML documents and its respective DTDs through SQL SELECT statement. Lastly, the transaction component performs database transaction services such as commit and rollback transactions during the data transfer to relational databases in order to maintain data integrity in both XML documents and relational databases.

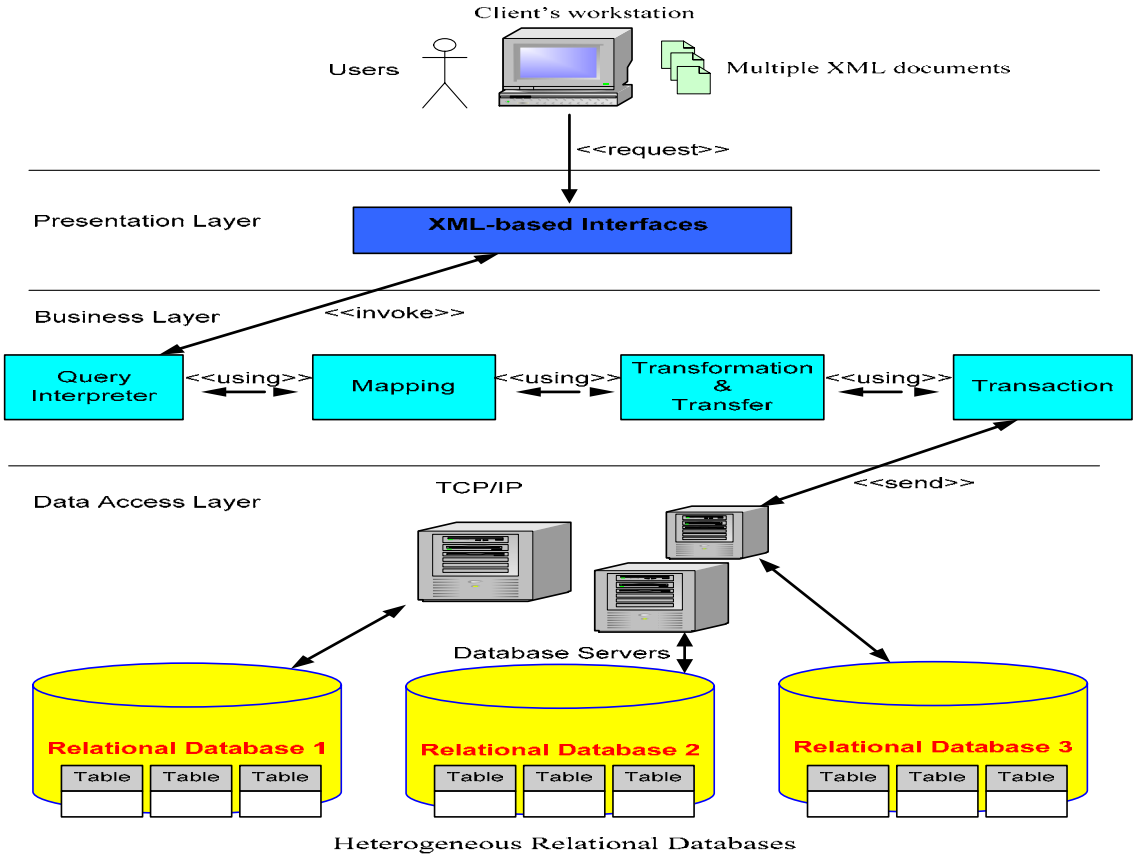

Fig. 2: JXDB System Architecture

4.0 JXDB ANALYSIS MODEL

The software analysis approach used in JXDB system is based on object-oriented analysis and modelled using Unified Modelling Language (UML). To model JXDB analysis process, we are using use case model driven approach. In fact, the main benefit of using this approach is that all design decisions can be traced back to user requirements, analysis, design, and implementation and testing. Hence, use case model driven approach can be implemented throughout the whole software development life cycle. Most software organisations have begun to adopt use case model to build their requirement model, where the software developers can understand what the users of the system wants and present these requirements back to the users because they also need to understand it from a developer's point of view. Fig. 3 illustrates the use case diagram for JXDB system. This figure models the system's requirements and what are the services that the user can do with the system. Examples of the scenarios where the system can offer its services to the users are load XML documents, insert, update, delete, and read XML data, etc, followed by transferring of these XML data from XML documents to relational databases and vice versa.

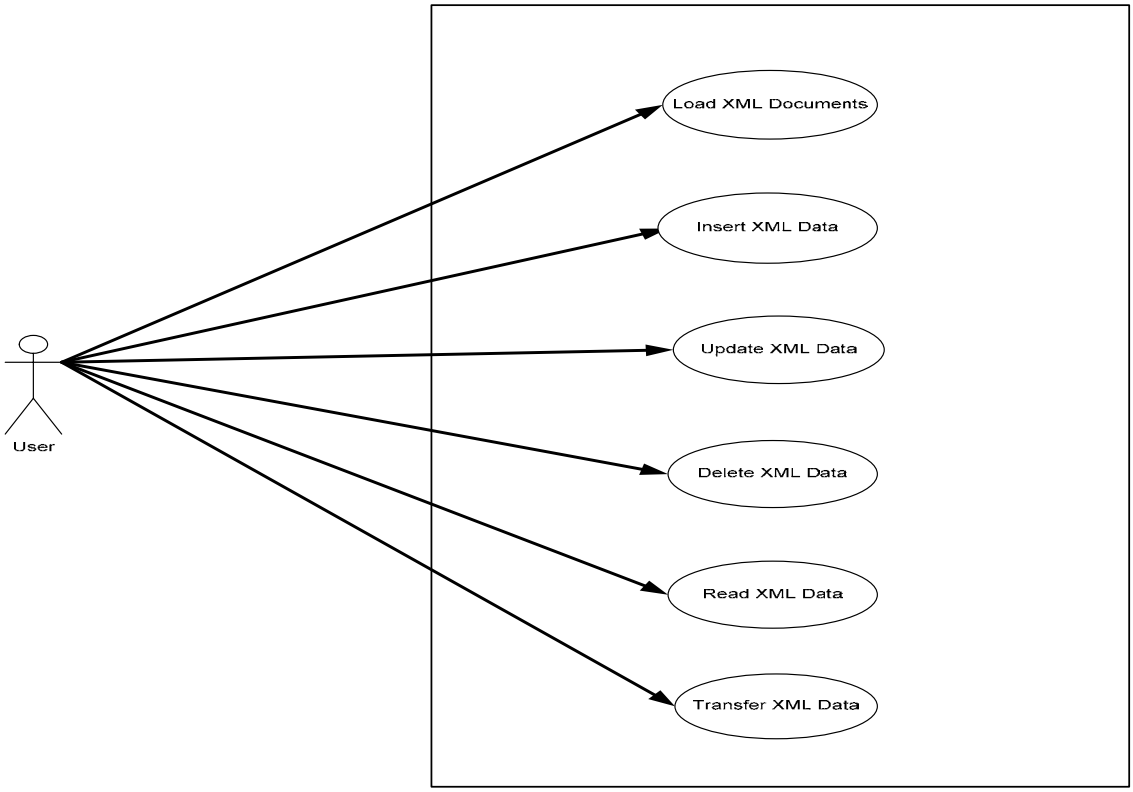

Fig. 3: Use Case Model of JXDB

Fig. 4 presents the object diagram for JXDB system, which can assist our analysis process by addressing the focus on what problem domain we are going to solve, without worrying the best way of how to solve the problem. In order to get the task done, these identified objects have to integrate with each other to provide the services needed by some other objects and, in turn, also need to use the services provided by other objects, so that they can complete the task. This is because each of these objects has been assigned specific tasks according to their roles in the system. Therefore, the diagram also helps to illustrate the relationships that exist between the identified objects in our system by linking each of them to related objects in their interaction. At this stage, all the objects shown in Fig. 4 are without their attributes, except the QueryInterpreter object because its attributes are important to define the layout of the system's user interface. The operations of each object are derived from the message calls occurred during the objects' interaction when carrying out the behaviour of the use case instances.

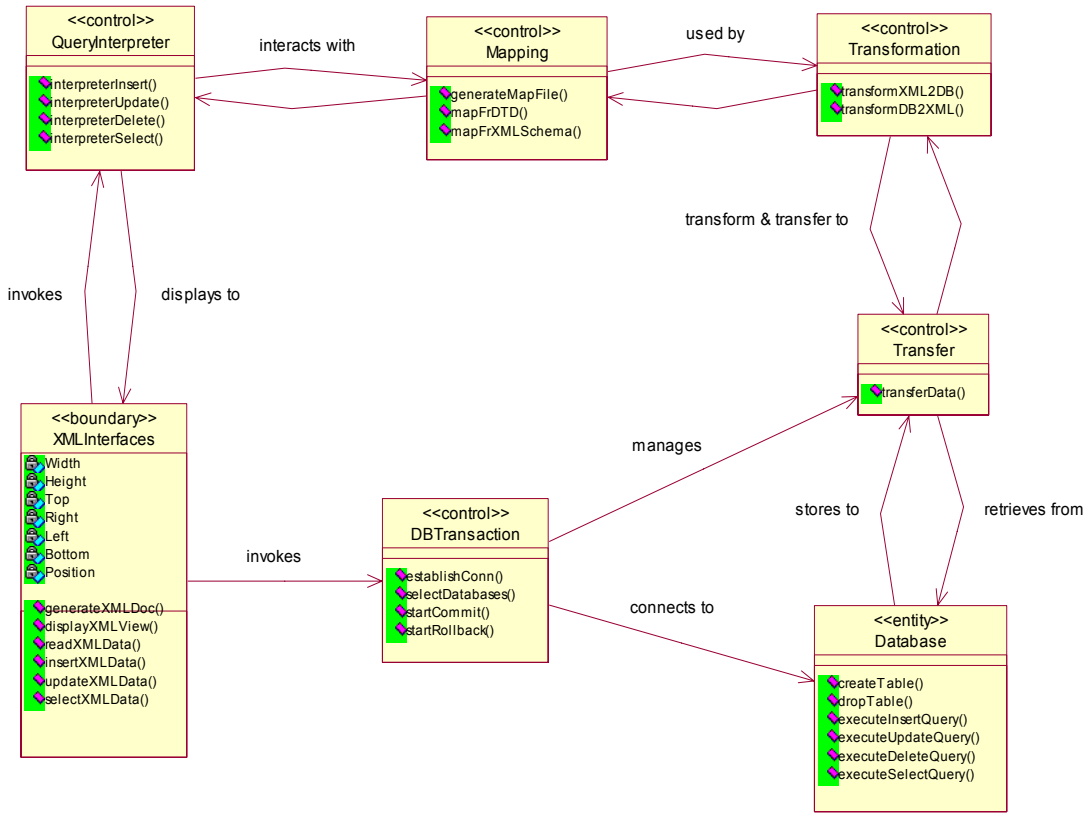

Fig. 4: JXDB Object Diagram

5.0 DATA TRANSFER STRATEGY

A common problem now in XML community is how to transfer and map XML data to and from databases. There are many different methods of transferring data but most are basically doing similar approach that is by mapping and conversion of data structure before transferring to databases. Moreover, XQuery still does not support direct updating of XML data to databases [6]. This section discusses briefly the data transfer strategy used in this JXDB system. In order to integrate XML data between XML documents and relational databases, we have to do some mapping on the XML structure defined in XML schema, such as DTD or XML schema, to the database structure defined in database schema [9]. The transformation and transfer component in this case is built on top of the mapping component. To transfer data from multiple XML documents to relational databases, there are few steps involved. First, the Query Interpreter component uses XQuery where the user has to enter the XQuery expressions to query and retrieve XML data from multiple XML documents, and then it will generate a new virtual XML result [3, 4, 11]. Some of the XQuery expressions that are supported here like Path expression, Element Constructors and FLWR expressions [1, 2]. Subsequently, it will display and model the virtual XML results on the data grid control and also it will also create DOM objects in memory for further manipulation of the XML results. From this data grid control, the user can insert, update or delete the in-memory XML results before mapping and transferring the data to relational databases. After that, the mapping component will perform the necessary mapping from XML schema or DTD to database schema based on the generated map file. Here, the user has an option to save a copy of XML results in a physical XML file. Once the mapping is in place, the Transform component will transform the XML data to Java objects before invoking Transfer component to transfer the data to the back-end relational database. At the same time, the system also generates a SQL script file when transferring to relational databases if we need to create new tables or drop existing tables. The DBTransaction component will manage the whole transaction during the data transferring process, and it is also responsible to establish and manage connection pooling to databases.

There are two commonly used mapping strategies to model XML data in XML documents; they are table-based mapping and object-relational mapping or object-based mapping [9]. We need to do a mapping from its DTD to database schema before transferring the data between XML documents and relational databases because the XML structure is not the same as database structure [3, 4, 9, 11]. In fact, both mappings are bidirectional meaning that they can be used to transfer data both from XML documents to the databases and from the database to XML documents. We have studied the pros and cons of both mapping strategies, and decided to use the table-based mappings, though it only works with a limited subset of XML documents depending on the application requirements. Despite this, this mapping satisfies our requirements even though there are some tradeoffs in return.

On the other hand, to transfer data from relational databases to XML documents, the implementation strategy is almost similar. First, users will issue a SQL SELECT statement to the database to select required data from a table or multiple tables using a JOIN statement, and then download the result sets to the client's side and generate virtual XML views using DOM API that were transformed from persistent Java objects. Subsequently, the system also generates its respective DTD and optional mapping file when it has downloaded the result sets to the client's side because these files might be needed later on in the mapping process. The generated virtual XML view is presented back to the users via the data grid control. Here, the user can continue to manipulate the data like before, such as insert, update or delete, before going through the same process of mapping and transferring the modified data back to relational databases for updating. Fig. 5 shows the graphical overview of JXDB's data transfer strategy.

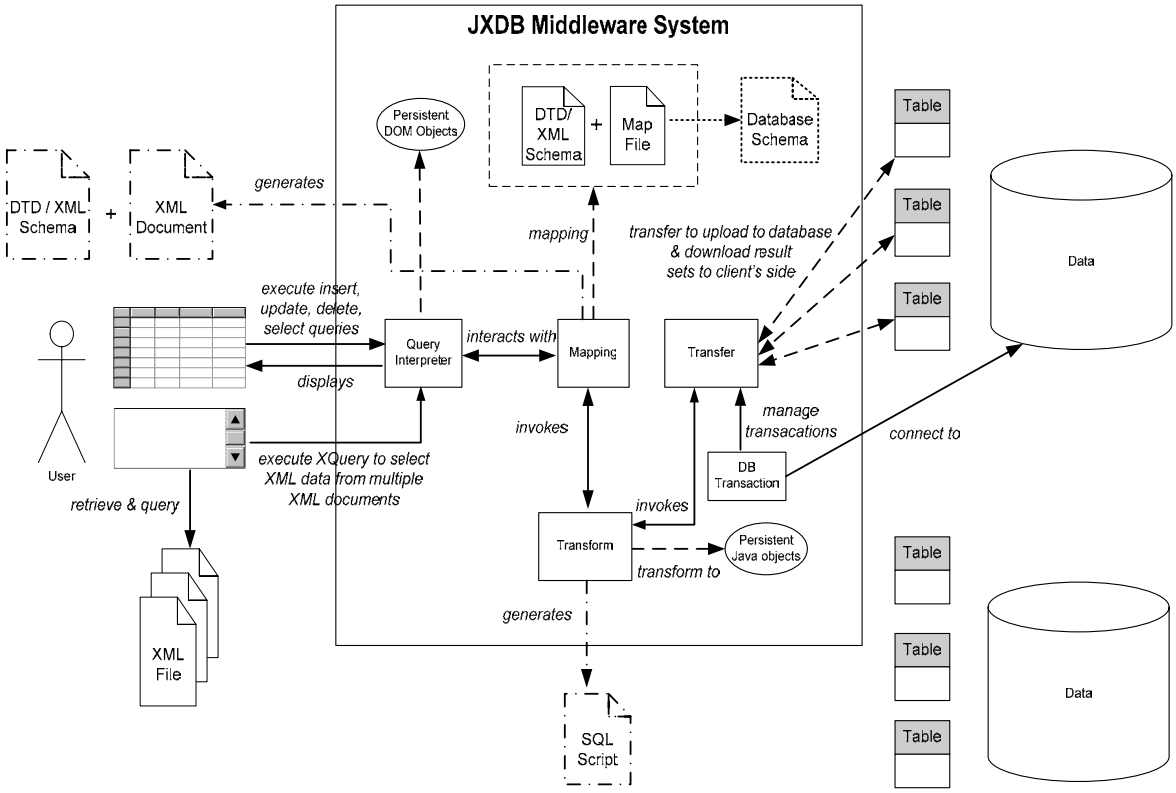

Fig. 5: JXDB's Data Transfer Strategy

6.0 CRITICAL REMARKS

We have analysed several critical remarks found in this system. Some of the remarks are due to the immature capabilities and facilities offered by W3C XQuery, which is the first query language designed for XML data to receive industry-wide attention and support from W3C. At the time this paper is written, XQuery is still being developed by the W3C XML Query Working Group and has obtained a "Working Draft" status [1, 2, 5]. Unfortunately, XQuery provides retrieval and query of XML data as read-only and it still does not provide full text search, insert, update and delete facilities for XML data [6]. Hence, the proposed system does not use direct insert, update and delete on XML data using XQuery, except for querying and retrieving of XML data from multiple XML

documents. Instead, we use a data grid control to generate a XML-based view over the XML document where users can directly insert, update and delete the data through the data grid control. Moreover, the inconsistency format between the XML and XQuery has made the implementation of XQuery as the query language for XML data rather tedious and complicated. Despite the goal of W3C to make XQuery the default query language for XML, XQuery itself is still not in XML format [5]. Thus, W3C is trying to work on XML syntax for the XQuery semantics in their future release working drafts. Therefore, in future it is possible that these XQuery limitations can be resolved once XQuery has been fully matured and has reached the W3C's Last Call Working Draft status before it fully becomes a W3C Recommendation [1, 5].

Presently, the system does not cater for processing a large volume of XML documents as it is using DOM APIs for processing and parsing XML. This is due to the weakness of XML DOM parser as DOM parser has several serious problems that will affect the performance-sensitive of an application. One of DOM problems is the fact that it loads the entire XML document into computer memory using much more memory. For example, to parse or query large XML documents on low hardware specification will be a bottleneck for any application. To overcome this, if it is related to hardware issue, then we can upgrade the hardware specification such as upgrade the system memory. Another subtle problem of DOM API is that the source code written for it must scan the XML document twice when parsing. Despite these, some of these problems could be addressed with a better underlying data structure designed to internally represent the DOM object model. However, it will be ideal that W3C works to improve on the performance processing of XML DOM parser. Instead, if it is related to software problems, we believe that by using SAX parser, it will be able to process a larger volume of XML documents and faster compared to XML DOM parser. This is because SAX API does not have a generic object model, so it does not have the memory or performance problems like DOM API. In this case, one consequence of using table-based mappings is that it only works with a limited subset of XML documents.

Another issue is that using middleware approach introduces another layer in the application's system architecture. In fact, it decreases the overall performance of the application. For example, instead of directly connecting to the back-end databases and let the XML-enabled databases do the necessary processing, it needs to do object-relational mapping and transformation of the XML data to relational schema every time they transfer the data to the database and vice versa. The issue gets more complicated if the XML schema and/or map file for the XML are changed frequently. Another similar subtle issue is that if there are lots of different types of structure of XML files and frequently new types of XML are being introduced that will definitely make mapping and transformation of schema more complicated. Hence, it will lead to a change in the mapping definition file and corresponding change in the relational schema. All these will incur much higher deployment and maintenance cost and vulnerable to more points of failure. In addition, sometimes this approach is not preferred by software developers as it increases complexity of the system.

7.0 FUTURE EHANCEMENT

There are several potential features identified, that can be added to this system to enhance and extent the XML-based middleware functionalities to offer better and richer features to its users. One of the potential future enhancement works is to web-enable and offer as web services over Internet, so that the users can access the system across the enterprise network at anytime and anywhere. This is the most significance feature because nowadays web services are also receiving great interest and supports by major software companies such as Microsoft, IBM, Sun, etc. A web service can publish and enable its programmatic application interface accessible by any remote applications over the Internet. Another potential future work that has been identified is to extent its ability to integrate with different types of databases other than relational databases, such as native XML databases and object-oriented databases. This will enable the system to be vendor and platform independent, and robustness so that it is capable to integrate with any types of back-end databases.

Though it may seem that a lot of future work has to be done, this middleware system is still a very promising initiative system for integrating XML data and heterogeneous relational databases in updating and retrieving of XML data to relational databases and vice versa. Moreover, it can provide a generic framework for software vendors to extent this framework to work on improving and extending the functionalities of this system. As the emerging XQuery becomes mature and recognised as the standard XML query language by W3C, more value-added facilities should be made available to the large XML communities, such as direct update and deletion of XML data using XQuery without a schema. This is the area that needs more work to be done to test using XQuery once it can provide direct inserting and updating of XML data. To use these with relational databases, the data in the database must be modelled virtually in XML format, thereby allowing queries over these virtual XML documents. However, this

situation will change when XQuery is finalised by W3C, as some major database vendors are already working on their own implementations. Another possible future work is to provide a generic mapping framework that can map any types of XML documents to relational structure without using an explicit map file or a schema. Most importantly, XML middleware products are relatively new compared to XML-enabled relational databases, thus scalability and reliability of this approach have not been fully tested.

8.0 CONCLUSION

In this paper a GUI-based XML middleware approach integrating XML with heterogeneous relational databases using object-oriented concepts are investigated. Various studies and tests were conducted on several XML middleware products, including both commercial and freeware products, that implemented similar approach in mapping and transferring of XML data to relational databases and vice versa. However, we only documented the study done on one of the existing XML middleware products, i.e. XML-DBMS version 1.0 that is used to transfer XML data to relational databases, and vice versa. These tests are used to analyse the strengths and critical remarks of these products, and based on the results, a conceptual system specification and design for the proposed JXDB is made by taking into consideration the pros and cons of these products. At the time this paper is written, the proposed JXDB is relatively one of the first few middleware to implement XQuery for querying and retrieving of XML data from multiple XML documents though it is still at its infancy. However, much work is going into solving many issues to improve its functionalities for it to evolve rapidly into a more mature standard query language like what SQL is as the standard query language for relational databases. Additionally, vast quantities of business critical data are currently stored in the existing back-end relational database systems. However, great demand is pushing for the ability to present that data in XML form to the client applications. Last but not least, XML middleware approach is suitable to be implemented in a distributed multi-tier application environment. This is because with this approach, connection pooling can be introduced to significantly improve the performance of these multi-tier applications, especially in applications that are not tightly coupled with the back-end databases or any existing legacy databases, and where the XML schemas and formats are not changed frequently.

REFERENCES

[1] Don Chamberlin, "XQuery: An XML query language". *IBM Systems Journal*. Vol. 41, No. 4, 2002.

[2] Don Chamberlin, Peter Fankhauser, Daniela Florescu, Massimo Marchiori and Jonathan Robie, "XML Query Use Cases"*, W3C Working Draft,* 15 November 2002. http://www.w3.org/TR/xmlquery-use-cases/.

[3] Eisenberg, Andrew and Melton, Jim, "SQL/XML and the SQLX Informal Group of Companies". *ACM Special Interest Group on Management of Data (SIGMOD) Record*. Vol. 30, No. 3, Sept. 2001, pp. 105 - 108.

[4] Eisenberg, Andrew and Melton, Jim, "SQL/XML is Making Good". *ACM Special Interest Group on Management of Data (SIGMOD) Record*. Vol. 31, No. 2, June 2002, pp. 101 - 108.

[5] Melton, Jim and Eisenberg, Andrew, "An Early Look at XQuery". *ACM Special Interest Group on Management of Data (SIGMOD) Record*, Vol. 31, No. 4, December 2002, pp. 113 - 120.

[6] Igor Tatarinov, Zachary G. Ives, Alon Y. Halevy and Daniel S. Weld, "Updating XML", *ACM Special Interest Group on Management of Data (SIGMOD) Record*. Department of Computer Science and Engineering, University of Washington. Vol. 30, No. 2, June 2001, pp. 413 - 424.

[7] J. Shanmugasundaram, H.Gang, K. Tufte, C. Zhang, D. J. DeWitt, and J. F. Naughton, "Relational Databases for Querying XML Documents: Limitations and Opportunities", in *Proceedings of the Very Large Data Bases (VLDB) International Conference*, 1999*,* pp. 302 - 304.

[8] J. Shanmugasundaram, E. Shekita, R. Barr, M. Carey, B. R. B. Lindsay, and H. Pirahesh, "Efficiently Publishing Relational Data as XML Documents", in *Proceedings of the Very Large Data Bases (VLDB) International Conference*, 2000.

[9] Bourret, Ronald. *Mapping DTDs to Databases, 2001*. http://www.xml.com/lpt/a/2001/05/09/dtdtodbs.html.

[10] Bourret, Ronald. *XML Database Products : Middleware*, 2000 – 2002. http://www.rpbourret.com/xml/ProdsMiddleware.htm

[11] Funderburk, J.E. , Malaika, S. , Reinwald, B, "XML Programming with SQL/XML and XQuery". *IBM Systems Journal*. Vol. 41, No. 4, 2002.

BIOGRAPHY

Lee Ching Kum obtained her Master of Software Engineering from Faculty of Computer Science and Information Technology, University of Malaya, Malaysia in 2003. Currently, she is working as a software developer in the private sector.

Sai Peck Lee is currently an associate professor at Faculty of Computer Science & Information Technology, University of Malaya. She obtained her Master of Computer Science from University of Malaya in August 1990, her Diplôme d'Études Approfondies (D. E. A.) in Computer Science from University of Pierre et Marie Curie (Paris VI) in July 1991 and her Ph.D. degree in Computer Science from University of Panthéon-Sorbonne (Paris I) in July 1994. Her current research interests include Software Engineering, Object-Oriented (OO) Methodology, Software Reuse and Framework-based Development, Information Systems and Database Engineering, OO Analysis and Design for E-Commerce Applications and Auction Protocols. She has published an academic book and more than 70 papers in various local and international conferences and journals. She had also served as the executive editor of a journal for 2 years, and has been in the reviewer committees and programme committees of several local and international conferences.