CHAPTER 6  FUTURE WORK AND CONCLUSION

This chapter is the concluding chapter for the whole research of this dissertation. It verifies the system objectives to determine whether it has achieved these objectives by evaluating its strengths or significance contributions to its users. It also addresses the limitations issues found in the system and their respective solutions. Moreover, as there is some room for future work and enhancement, this chapter gives significant recommendations for the proposal of future work for the JXDB system. Most importantly, these proposed future features are beyond the scope of this dissertation and also due to the limited time constraint of this project development life cycle. Therefore, these potential features that have been put as the future implementation of a more comprehensive XML-based middleware framework.

6.1 Significance of Research

This section describes briefly the significance of this dissertation research. It will help to evaluate the achievement of JXDB system's objectives as presented in Chapter 1. One of the research achievements is to contribute significant benefits to commercial companies such as small or medium-sized companies that make use of small scale of XML data at the front-end and relational database management system at the back-end for their XML requirements. Furthermore, it is also beneficial to the software development of XML and relational database communities. Below are several significances of this research:

- Any user who is without any in-depth programming of XML and SQL skills can use JXDB system to manipulate, update and transfer data between XML documents and heterogeneous relational databases. This is because the system provides an ease-of-use
of graphical user interface for users to update and transfer the XML data to and from the underlying back-end relational databases.

- Some of the significant features provided by using JXDB are the abilities to query and generate virtual XML views from these underlying different and distributed non-XML data that is stored in a variety of repository formats such as relational structures on heterogeneous relational databases, and as well as the future enhancement ability to extent the framework to cater for object structures on object-oriented databases. This is important because most often users desire only a subset of an XML view’s data. Moreover, often users need to synthesize and extract data from multiple tables or views. This can be done by mapping and transferring data from the underlying relational database system to compatible XML schemas at the client’s side in order to generate the default XML views and display the data via a data grid control. Apart from that, once the data is downloaded to the client’s desktop via Java data access objects in memory, users can then create application-specific XML views on top of these default XML views. Furthermore, these application-specific views can be manipulated using XQuery, a general-purpose query language for XML for providing updating facilities against these virtual XML views. In JXDB, these queries are specified using the same language that is used to specify XML views, namely XQuery. Subsequently, JXDB is able to execute these queries efficiently by performing XML view composition so that only the desired relational data items are materialized in the process. In summary, JXDB provides a general means to publish and query XML views of existing relational data. Users always use the same declarative XQuery query language, regardless of whether they are creating XML views of relational data or querying those XML views.
Similarly, JXDB also provides the ability to query and create virtual XML views from multiple XML documents using XQuery.

- Currently, facilities like updating XQuery queries, which include inserting and deleting queries, are not designed to be part of the goals of XQuery version 1.0 specifications. However, by using this JXDB framework, it is able to provide simple parameterised approach for users to manipulate updating facilities against the XML data such as to insert, update and delete queries prior to the process of transferring the XML data to back-end databases.

- Moreover, JXDB is designed to provide lower development cost and reusable components. Hence, it is best suitable for users of small or medium sized companies whereby they require exchanging critical transaction data with their business partners using XML technologies while still leveraging existing relational database technology. Normally, in the production environments of these companies, the actual processing volume ranges from small to medium scale volumes. They often need to use XML data at the front-end whereas at the back-end still uses existing relational databases. Therefore, they need a XML middleware to transfer and update these XML data to their existing back-end database systems. In fact, they do not wish to spend any extra cost in buying expensive XML solutions or even outsource the solution to other IT integration company to develop the system because this will incur unnecessary cost in their budget.

- On the other hand, it provides an extensive framework for software developers to further extent JXDB system’s capabilities. For example, to implement the framework as a set of public accessible APIs where it can be called to perform update and transfer tasks automatically from an external application. Hence, it is more flexible and robust for any existing application to integrate with JXDB to perform manipulation of XML
data before transferring them for storage in relational databases. In addition, JXDB has been customised to include additional data access adapters that provide database vendor independent whereby any application can easily integrate and transfer XML data to any type of relational database without having to re-compile and re-code the applications if there is any requirement to change the back-end database systems. Using this, XML middleware framework approach gives software developers the ability to deliver structured data from a wide variety of applications or data sources to clients' desktops for local processing and presentation using virtual XML views.

- **XML middleware** is an independent server application or a set of APIs that resides between the front-end user XML application and the back-end databases to transfer XML data between them. One of its benefits is that it is able to provide services to a large scale of applications connected over the TCP/IP network. Subsequently, it reduces the complexity of application integration in a multi-tier enterprise solution. This is because XML middleware approach permits applications to be divided into multiple layers allowing abstraction and reducing application complexity at each layer. Thus, multiple applications and users can connect and share one XML middleware over the connected network.

### 6.2 Critical Remarks

We have analysed several critical remarks found in this system. Some of the remarks are closely 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. Others query languages that were developed earlier include XQL,
XML-QL and Quilt. At the time this dissertation is written, XQuery is currently being developed by the W3C XML Query Working Group and has obtained a "Working Draft" status (Jim and Andrew, 2002). At this stage, the work on XQuery is still in progress, and as the drafts evolve it may be updated, replaced or made obsolete by other documents at any time. Apart from these, there are also many inconsistencies within and between the W3C working draft documents (Jim and Andrew, 2002). Currently, 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 (Igor et al, 2001). Any full text search facilities provided by different XML software vendors are not standard and proprietary to their own implementation approaches. Thus, this JXDB system does not implement direct XQuery updating facilities such as direct insert, update and delete on XML data, except for querying and retrieving of XML data from multiple XML documents. Instead, a data grid control is used to generate 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 (Jim and Andrew, 2002). However, W3C is trying to work on XML syntaxes for the XQuery semantics in their future release working drafts. Therefore, in the future it is possible that these XQuery's limitations can be resolved once the XQuery has been fully matured and has reached the W3C's Last Call Working Draft status before it becomes fully a W3C Recommendation (Chamberlin, 2002).
Presently, JXDB 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 DOM API problem is that the source code written for it must scan the XML document twice when parsing the XML. Despite these, some of these problems could be addressed with a better underlying data structure design to internally represent the DOM object model. However, it will be ideal that the W3C will work 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 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.

Another issue of using middleware approach introduces another layer in the application’s system architecture. Thus, it decreases the overall performance of the application. For example, instead of allowing the XML-enabled databases to do the necessary processing, instead they need 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. It 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. Sometimes this approach is not preferred by software developers as it increases complexity of the system.

6.3 Future Work

Upon the completion of this JXDB prototype, there are several potential features, which have been identified, that can be added to this system to further 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 the Internet, so that users can access the system anytime and anywhere. This is the most significance feature because nowadays web services are also receiving great interests and supports from major software companies such as Microsoft, IBM, Sun and etc. Web services have been hyped as the next big thing on the front of Internet-based enterprise business applications. It enables different applications running on different platforms and technologies to be able to interact and exchange message with each other using web standards. A web service can publish and enable its programmatic application interface accessible by any remote application over the Internet. Once it is deployed as a web service on the web server, a remote user can invoke its exposed web methods via the Simple Object Access Protocol (SOAP) over HTTP.
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 the native XML databases and object-oriented databases. This will surely enable the JXDB framework to be vendor independent and platform independent. As a result, it is a robust and extensive framework because its capability to integrate with any type of back-end databases. Currently, JXDB has been tested using different relational database products such as Oracle 9i, Microsoft SQL 2000, MySQL and Access 2000 via its reusable data access adapter components. Besides relational database systems, JXDB’s framework can customise and add additional data access adapters to its data access subsystems to access heterogeneous databases such as object-oriented and native databases. For example, these few years, object-oriented databases have been gaining positive responses from most software industry. Hence, JXDB system can further be enhanced to integrate with an object-oriented database, such as POET, in order to obtain improved and smooth performance. This is because by migrating to this type of object-oriented database is believed to have better performance than the traditional relational database with its effective implementation of Object Query Language (OQL), especially for an object-oriented application framework. However, the full potential of an object-oriented database has yet to be realized.

Though it may seem that a lot of future work has to be done, however this middleware system is still a very promising initiative system in 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. On the other hand, another major change in the coding would be to implement JXDB using the SAX model instead of DOM in order to streamline the processing of the
XML files. The event-driven model of SAX as described earlier, allows for faster parsing and processing with smaller memory requirements. Hence, this would enable JXDB to process larger XML files at a faster processing rate. Overall, JXDB is a good preliminary start in implementing an XQuery processor for integrating XML with relational databases, but has a long way to go before W3C can offer a complete and stable implementation of XQuery. Unfortunately, the current XQuery implementation is still in its early development and standardization phases, so a true full implementation may not be possible until then. Nevertheless, in the near future, when the emerging XQuery become mature and recognised as the standard XML query language by W3C, then 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 once 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 type 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.
6.4 Conclusion

In conclusion, XML is an extremely versatile markup language, self-describing, capable of labelling the information content of variety of data sources, such as semi-structured and structured documents, relational databases, and object repositories. Presently, XML has emerged rapidly as the leading language for data exchange and content management as it is largely implemented not only on the Web and between business partners, but also within the organization itself. The deliverable of this dissertation is a working prototype of GUI XML-based middleware interface integrating with heterogeneous relational databases, which is known as JXDB middleware system. Apart from that, this dissertation report provides studies on selected existing XML middleware and XML-enabled databases on the areas where most of them are lacking of these elements, such as system interactivity, vendor dependent and proprietary that is not open standard. This is because these middleware and XML-enabled databases consist of rather static and programmatic APIs interface that is insufficient in providing interactive exploration for non-software developers to use. In this case, JXDB system provides an ease-of-use graphical user interface to import and export XML data from multiple XML documents to relational databases and vice versa. Java has been identified as the most suitable technology to be used as the programming language to develop this system.

This dissertation report describes the overview system architecture of this graphical user interface XML-based middleware, which is a unique tool for integrating and querying XML data from XML documents to back-end relational databases, by marshalling and un-marshalling these XML data to and from heterogeneous relational databases. Here, it describes the object-oriented system architecture used and focuses on the XML mapping
and transformation technologies to marshal and un-marshall the XML data to and from heterogeneous database systems. The implementation of this system has adopted XML middleware approach. In this architecture, the XML-based middleware interface provides a uniform graphical user interface to query and manipulate the XML views from heterogeneous information sources. This XML-based interface provides local views of data sources in a standard data model. For example, the SQL query result sets from the database server can be presented as virtual XML views locally via disconnected record sets, without incurring any overhead to the database server. However, these local views can be queried using XML query language, which is XQuery, in a limited way according to its capabilities. Though XQuery, which is emerging rapidly as the standard query language for XML, has been chosen to be implemented as the query language at the front-end for users to query natively the XML data from XML documents before they are transferred to heterogeneous databases for later retrieval and storage, however XQuery updates were not discussed thoroughly here as it is beyond the scope of this dissertation and still hold W3C working draft status. Apart from that, it was chosen to demonstrate the promising future potential functionalities of XML query language for XML data, as what SQL is for relational data.

This dissertation also performed thorough analysis and experimental of using several different types of relational databases, namely Microsoft SQL 2000, Oracle, and MySQL database tested here. Finally, the experimental results demonstrated that the performance advantage gained by using this XML middleware approach can benefit most enterprises especially small and medium enterprises, whereby now they are able to integrate XML to their existing back-end relational database systems. Large enterprises can benefit by extending this framework and provide more promising values-added services to it. Last but not least, with the widespread adoption of XML technologies as a popular data exchange
format integrating with wide diverse of databases, there are likely large interest in developing and growing its capabilities until it can rapidly gain high level of adoption across all industry of XML applications and usages.