CHAPTER 3  RESEARCH METHODOLOGY

This chapter will describe the methodology for the whole life cycle of this project. In general, a methodology is known as a method that is defined to produce a systematic and orderly approach to achieve predefined objectives and goals of a project. Thus, it is essential to perform an appropriate methodology before proceeding to analysis, design and implementation of the proposed working system. The methodology used in this project consists of a set of phases in chronological sequences. Every phase is essential and must be completed before proceeding to the next phase. Every phase specifies all the steps, which are required in order to achieve the objectives of this project. Here, the methodology starts with specifying the overall framework on the development of this project, which is elaborated in the next section.

3.1  Project Framework

The project framework is the primary component that explains the methodology used in this project. This overall framework has been customised to suit to this dissertation. It consists of five phases, namely feasibility review, system design and analysis, implementation, testing and evaluation, and finally proposed future work of the system. In every phase, it has a set of sub-ordinate objectives to be satisfied in the whole software life cycle, which will lead towards achieving the project’s objectives.
Table 3-1 briefly explains the overall project framework that will be conducted accordingly in the following chapters.

<table>
<thead>
<tr>
<th><strong>Project Phase</strong></th>
<th><strong>Objectives</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Feasibility Review</td>
<td>To study and review existing the XML technologies used in XML middleware and XML-enabled databases To study and identify suitable enabling technologies to be used in the object-oriented software development</td>
</tr>
<tr>
<td>System Analysis and Design</td>
<td>To identify and specify the system analysis and design models using object-oriented and RUP methodologies To implement UML modelling and notation to draw use case and class diagrams identified during object-oriented analysis and design phase</td>
</tr>
<tr>
<td>Implementation</td>
<td>To develop a working prototype based on the predefined system designs and specifications</td>
</tr>
<tr>
<td>Testing and Evaluation</td>
<td>To test the system to minimize system error or bug To verify whether the system has satisfied the predefined designs and specifications To test the prototype with several major relational databases, like Microsoft SQL 2000, Oracle, and MySQL. To evaluate whether the technologies implemented in this system satisfy the overall objectives of this project</td>
</tr>
<tr>
<td>Enhancement Work</td>
<td>To implement a unified framework approach for updating and storing arbitrary XML documents taking into consideration backward compatibility To extent its functionality as web services over the Internet</td>
</tr>
</tbody>
</table>

### 3.2 Feasibility Review

In this section, the main objective of this feasibility review is to study and review the technologies used in existing XML middleware and XML-enabled databases and draw a comparison between the current systems and the proposed system. These reviews are
gathered from related web sites, product reviews, online XML resource guides and online XML magazines. It gives a summary of these tools and their features. After analysing the pros and cons of XML middleware and XML-enabled databases, the analysis results will then be used to construct an appropriate design for the proposed XML-based middleware interface in integrating with heterogeneous relational databases by incorporating new features and technologies. In fact, it is hoped that with this, it can help to develop best-of-breed solution. To start off this review, the first stage begins with gathering and finding information about the approaches and strategies used by the existing XML middleware and XML-enabled databases and then draw a comparison between the current systems and the proposed system.

3.2.1 Review of XML Middleware

This review looks at the aspects of the available XML middleware in the present market, which are designed to be data transport to marshal and un-marshal data between XML documents and relational databases developed by third party software vendors. Middleware is the software used by data-centric applications to transfer XML data between XML documents and databases. It is usually lightweight, and usually runs in the same process space as the application (Bourret, 2003). Here, these middleware use database drivers such as ODBC, JDBC, or OLE DB to connect to relational databases to access data.

The following sections describe several examples of popular XML middleware that are used as a tool to map XML documents to relational tables, and vice versa, in order to transfer data between XML documents and relational databases in both directions. Those XML middleware that offer only one way of data transfer, either from XML to relational
databases or the other way round, will not be discussed here. A generic graphical view of the architecture of middleware is shown in Figure 3-1.

![Figure 3-1: Generic Graphical View of Middleware Architecture](image)

3.2.1.1 Allora

One of the most popular middleware among XML software developers and software vendors is Allora. Allora is a commercial product developed by HiT Software Inc. It is a middleware that can be invoked from any application to transfer data between an XML document and a relational database, either from a XML document to relational database or the other way round. According to its product description, Allora can be used in three different ways where it can:

- Directly transfer data between an XML document or DOM tree and the database.
- Be called to transfer data between an XML document and the database through SOAP interface.
- Be used as an XML data binding engine in any integration between XML and database applications. This is where the software developers generate classes which are specific to the data in an XML document or database tables. With these classes generated, the software developers can use them directly in coding their applications by populating

---

32
these classes from the XML document or relational tables, and thus perform the transfer of data to an XML document or relational tables.

In addition, Allora implements object-relational mapping definitions with extra capabilities, such as the mapping provided is able to use JavaScript or VBScript to convert the data when extracting it from the database, which is not reversible and so on. However, these additional capabilities which are not related to the scope of this dissertation will not be described further. One of its useful mapping technologies used is its flexibility to be parameterised in programmatic codes.

3.2.1.2 XML-DBMS

Another middleware that is to be discussed here is XML-DBMS. It is also a middleware that is used to transfer data between an XML document and a relational database. It also uses an object-relational mapping that is described by an XML-based mapping language. Here, the mappings can be written manually from scratch or generated automatically from a given DTD or database schema. 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 object-relational mapping languages, it uses a language that is able to map complex element types to tables, maps simple element types, attributes, and PCDATA to columns and maps nesting to joins. In addition, there are some useful features like performing simple transformations in an XML document and distribute the data from a single document across to multiple databases. As usual, the additional features which are not in the scope of this dissertation will not be covered here.
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, the software developers 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, the software developers can use filters like the “where” expression to specify only the specific data to be retrieved. Thus it provides parameterised approach in this case for greater flexibility to the software developers. Moreover, data can be deleted from the database according to a particular filter and map. In spite of many useful features that this middleware offers, it is just a set of Java packages used by software developers to transfer data between XML documents and relational databases in their applications. It can be run from the command line or a program to transfer data between an XML document and a database.

3.2.1.3 Comparison of Existing XML Middleware and the Proposed System

From the above explanation on the existing XML middleware, a summarised comparison between these middleware and the proposed system can be concluded from these reviews. In these comparisons, brief explanations on the advantages and disadvantages of each of these system designs and functionality will be documented here. As a result, a conceptual system specification for the proposed system is made by taking into consideration the pros and cons of the overall system and functionalities.
Based on the limitations faced by most of the existing XML middleware, the following conclusions have been drawn to design the proposed system:

- This proposed system should be able to provide a simple and yet user-friendly graphical XML-based interface to transfer XML data from XML documents to relational tables, and vice versa, to be used by any users of the system. Most of the existing middleware or data transport doing import and export of XML data are just sets of Java packages such as APIs, Java classes, or even Java objects that are available for any software developers who wish to transfer data between an XML document and relational tables. However, some do provide a simple graphical user interface but only for the mapping processes of DTDs or XML schemas to relational schemas and vice versa. This is because the mapping process is a tedious process which might involve complex element types, user-defined types, relationships, and etc.

- It also would have the ability to establish connections to multiple heterogeneous relational databases simultaneously. So far, this feature has not been catered by any XML middleware or XML-enabled database vendors, even though most of the middleware can integrate with popular relational databases like Microsoft SQL, Oracle, MySQL, etc. However, their connections are not concurrently, but one type of database connection at one time.

- Additionally, unlike most of the XML middleware, the proposed system has the capability to create new table(s), drop existing table(s), and query column and table attributes. This process is displayed using a virtual XML view over the relational databases.

- Another feature of the proposed system is to support database transactions like two phase commit and rollback transactions for any XML data transfer from XML
documents to relational tables for actions like insert, update or delete actions. This is to ensure data integrity during the data transfer from XML documents to the relational tables.

- Unlike most existing middleware, this proposed system also provides support for XQuery features when extracting XML data from multiple XML documents. So far, most of them only provide the ability to extract data from a single XML document. In addition, a lot of these middleware are still using proprietary approaches like using a join statement to extract and join XML data from multiple XML documents. However, some are already moving towards providing this feature using W3C standard XQuery to perform the task as there are more growing needs to extract data from multiple XML documents as XML information gets larger and more complex.

- Lastly, like most of these middleware, the proposed system also provides support for SQL queries and actions like by using a select statement to extract specific XML data, insert, update and delete XML data in relational tables through an XML-based and GUI interface.
Table 3-2 shows summarised findings that led to the conclusions made in designing this XML-based interface system. Moreover, this system will incorporate the required basic concepts from the existing XML middleware into this proposed system.

<table>
<thead>
<tr>
<th>XML Middleware Specifications</th>
<th>Proposed System Specifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pre-defined sets of mapping documents or templates for mapping and data binding for different XML documents.</td>
<td>Allow flexibility of specifying mapping definitions via its built-in functions and parameterised approach for different XML documents.</td>
</tr>
<tr>
<td>Lack of user interactivity.</td>
<td>Provide interactivity using wizard-based approach.</td>
</tr>
<tr>
<td>Lack of user-friendly interfaces leading to complexity usage of data transfer APIs, classes, objects or COMs during application coding.</td>
<td>Provide well-designed and user-friendly interfaces on XML view of XML documents and relational tables. The latter is known as virtual XML view over relational databases.</td>
</tr>
<tr>
<td>Lack of two phase transaction features when performing transaction actions like insert, update or delete XML data in relational tables.</td>
<td>Provide two phase transaction features like commit and rollback during data transactions to relational tables.</td>
</tr>
<tr>
<td>Ineffective extraction of data from multiple XML documents using proprietary approach.</td>
<td>Provide XQuery which is W3C recommendation to extract and query from multiple XML documents.</td>
</tr>
<tr>
<td>Lack of features for connecting to different multiple relational databases concurrently.</td>
<td>Provide connection access to different multiple relational databases concurrently using JDBC connectivity.</td>
</tr>
</tbody>
</table>
3.2.2 Review of XML-Enabled Databases

Several major database vendors such as Microsoft SQL, IBM DB2, Sybase, and Oracle have not left behind the race to incorporate XML support into their database products. Currently, they are extending their products for data integration between XML and relational databases. Relational databases that have built-in XML supports are known as XML-enabled databases, which contain extensions for transferring data between XML documents and their own database structures. In order to solve the problem of cost effectiveness, convenience and automatic import and export of XML data between XML documents and their relational databases, these vendors have developed various tools that are able to perform automatic conversion and transformation of XML documents into relational tables. In the following section, out of the various database vendors' solutions available in the market, only two solutions will be discussed and reviewed here. Figure 3-2 shows the generic graphical view of XML-enabled databases.

Figure 3-2: Generic Graphical View of XML-Enabled Database Architecture
3.2.2.1 Oracle

Each of these solutions approaches the XML problem domain using different strategies but almost similar technology patterns. As for Oracle, Oracle XML SQL Utility models XML document elements as a collection of nested tables. These enclosed elements are modelled by employing the Oracle Object data type (Dayen, 2001). For example, the SQL-to-XML conversion constructs an XML document by using a one-to-one association between a table, referenced by Object data type, and a nested element. The XML-to-SQL might require either data model amending by converting it from relational into object-relational model or restructuring the original XML documents. Oracle offers XML software developers five different APIs kits for different programming languages; they are Java, C, C++, JavaBeans and PL/SQL. According to Oracle description, their XML SQL Utility, known more as XSU, enables software developers to:

- Transform data retrieved from object-relational database tables or views into XML.
- Extract data from an XML document and, using a canonical mapping, insert the data into the appropriate columns/attributes of a table or a view.
- Extract data from an XML document, and apply this data to the appropriate columns or attributes for updating or deleting purposes.

On the other hand, even with all the rich-features of Oracle's XSU had to offer to its software developers, there are two main limitations:

- Limited mapping support. The default SQL mapping provided by XSU maps XML elements to table or query column names and values (Guardalben and Atre, 2002). However, it is usually not what a software developer wants because one cannot rename
columns, skip columns, or produce hierarchical XML documents. On the other hand, when using object-relational types, a software developer can create objects with complex containment configurations, and those objects can be serialised into complex, nested XML documents. Thus there are several drawbacks using this solution. Firstly, to XML-enable a DBMS, the DBMS itself has to change, and it is difficult to create complex relationships between XML elements/attributes and SQL expressions. Secondly, SQL object names have to match the names of XML elements/attributes; achieving that can be tedious, since XML names and SQL names follow different syntactical rules.

- Lack of graphical development tools. Oracle’s XSU provides only complex APIs or command-language tools to implement the data transfer, thus in order to use those APIs, one has to go through the learning curve (Guardalben and Atre, 2002). To simplify the generation and storage of complex XML documents, it is often desirable to have graphical tools that can assist the software developers to define mapping processes between SQL structures and XML elements or attributes.

3.2.2.2 Microsoft SQL

Microsoft’s objective is to make SQL Server 2000 a fully fledged, XML-enabled database server (Dayen, 2001). With the new XML features introduced with SQL Server 2000 support, software developers can use them to:

- Create XML Views using annotated XDR schemas.
- Use XPath to query XML Views.
- Retrieve and write XML data by means of XML extension to Microsoft Transact SQL.
The first two items provide supports for using XML queries rather than SQL queries as the prime query tools. The third item is a provided set of convenient XML extensions to the SQL language that is intended for traditional SQL developers. Actually, XDR schema syntax is Microsoft’s proprietary format for describing XML schemas. XDR schemas provide a richer set of data types and other features than the standard DTDs, and it is very similar to XML schemas, which is recommended by W3C (Dayen, 2001). The next release of SQL Server will most likely introduce the W3C XML Schema Standard (XSD) features. Therefore, the advantage is that queries to the database, when used with annotations to describe the mapping of XML elements and attributes to the database tables and fields, will return the results in the form of XML documents, or XML views. In this way, software developers can specify an XPath query against the XML view created by the XDR schema, where this XML view supports a large set of annotations. Moreover, they are not required to specify any annotation; but they are allowed to use a default mapping. Like Oracle and IBM’s default mapping support, the resulting XML is similar as well. On the other hand, XML Views can only be used to retrieve data (XML marshaling), and are read-only tools. As said before, SQL supports W3C XPath to query XML views as a standard query language for XML. However, the supported XPath for querying SQL Server is a subset of the full XPath standard because XML views generated by SQL Server are the ‘virtual’ XML documents and are not materialised in a stream or file.

As for retrieving and writing XML data to relational tables, Microsoft has introduced two basic methods for using extended SQL to operate on these XML data (Guardalben and Atre, 2002). The first method depends on an ad-hoc clause or known as FOR XML clause for the SELECT statement, and the second method is the Transact-SQL OPENXML function to insert data formatted as an XML document. The FOR XML clause of the
SELECT statement is used to return results as XML documents rather than standard relational data, whereby it can be executed in three XML modes, they are raw, auto and explicit. The only related mode here is the explicit mode thereby it provides most flexibility to software developers to specify the shape and names of the resulting XML document. Finally, as for the second method or Transact-SQL OPENXML, it provides a row set over an XML document. As such, an OPENXML is similar to a table or a view provider, whereby the software developer can move an XML document to database tables. An OPENXML can be used in SELECT or SELECT INTO statements.

From the explanation above, Microsoft’s SQL Server XML Support has shown great commitment to support new W3C XPath by tightly integrated XPath in the engine processors of SQL Server via its SQL extensions and powerful DBMS integrated XML processors (Guardalben and Atre, 2002). Even though Microsoft’s SQL Server XML Support offers a rich set of infrastructures, and in fact like other solutions, it still has some limitations as shown below:

- Only support Microsoft SQL Server 2000. This is because the overall XML support for Microsoft SQL Server is based on its proprietary extensions from annotated XDR or SQL extensions, which include FOR XML clause or OPENXML identifier. Thus it is impossible to extend to other DBMSs unless those data sources are embedded within a SQL server as heterogeneous linked servers.

- Limited support for mapping definitions. Though Microsoft provides flexible declarative-mapping definitions using XDR. It has two main drawbacks of a mapping tool. First, it does not provide instructions on what to do when database tables lack of
either unique indexes or primary keys. Second, it does not provide how to create tables on-the-fly during a decomposition operation.

3.2.2.3 Comparison of XML-Enabled Database and the Proposed System

After reviewing these popular XML-enabled relational databases’ efforts in incorporating XML supports in their products, a simple comparison of features between XML-enabled databases and the proposed system is summarised in Table 3-3.

<table>
<thead>
<tr>
<th>Vendor</th>
<th>Mapping Rules</th>
<th>Single / Multiple table(s)</th>
<th>Mean of Transformation</th>
<th>Directions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Oracle</td>
<td>Implicitly; by constructing object-relational data model</td>
<td>Multiple</td>
<td>Designated Java classes</td>
<td>XML -&gt; Database</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Database -&gt; XML</td>
</tr>
<tr>
<td>DB2</td>
<td>Data Access Definition file</td>
<td>Multiple</td>
<td>Designated stored procedures</td>
<td>XML -&gt; Database</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Database -&gt; XML</td>
</tr>
<tr>
<td>Microsoft SQL</td>
<td>SQL extension; row set function</td>
<td>Multiple for extraction; Single for storing</td>
<td>By using SQL construct FOR XML and row set OPENXML</td>
<td>XML -&gt; Database</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Database -&gt; XML</td>
</tr>
<tr>
<td>Proposed System</td>
<td>Declarative mapping approach</td>
<td>Multiple</td>
<td>Designated Java classes</td>
<td>XML -&gt; Database</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Database -&gt; XML</td>
</tr>
</tbody>
</table>
3.3 System Architectural View: 4+1

An architectural view of a system is the fundamental organisation of the system as a whole. A system's architecture has a number of main elements; they are static elements, dynamic elements, how these elements integrate to work together, and the overall architectural approach that guides the organisation of the system (Scott, 2001). In addition, software architecture is not a one-dimensional thing, it is made up of concurrent multiple views (Quatrani, 2000). With reference to Figure 3-3, it shows the different views of system architecture, which is known as 4+1 view of architecture. It also shows that Scenario is the view for all categories of users. It is also known as Use Case View where its goal is to provide understandability and usability by demonstrating and validating the logical, process, development and physical views. Logical View is the view of End-User where it addresses the functional requirements of the system and what service system provides to its users. The Development View shows the actual software module organisation and software management within the development environment. In short, it is a view for software developers to provide basic guidelines for the software developers to implement the system. Whereas, the Process View is a view focuses on the run-time implementation structure of the system. Usually, it is a view for the system integrator to show performance, reliability, scalability, integrity, system management of the system. Last but not least, Physical View shows the configuration of run-time processing elements and the software processes resided in the system. It is also known as Deployment View. It is a view for the system engineer to display the network topology, communication, and the layout of the hardware and software components.
The overall software development approach for this dissertation will be modelled using object-oriented development techniques, and where applicable, will integrate with some features of Rational Unified Process (RUP) methodology. Object-oriented techniques are chosen as they provide reusable components, system stability and ease of maintainability. As a result, significant levels of software reuse in the architecture and design of the proposed software systems can be produced, and thus leads to a smoother implementation and testing processes. Ultimately, it will produce a better and maintainable system. On the other hand, RUP is also chosen because it is an extensive set of guidelines for software development focusing on software analysis and design. Generally, RUP methodology can be summarised into the following features which will be implemented in the proposed system's development activities; they are iterative and incremental in nature, use-case driven and architecture-centric. RUP life cycle consists of four important phases:

- Inception. Goal: to specify the project vision, scope and primary functionality
- Elaboration. Goal: to capture the functional and non-functional requirements, and design the architecture
• Construction. Goal: to implement the system as a series of incremental iterations

• Transition. Goal: to deliver the system to user community and maintenance

For this purpose, UML is selected as the modelling language and notation used for modelling object-oriented and RUP methodologies. Unified Modeling Language (UML) is a language used to specify, visualise, and document the artifacts of an object-oriented system under development (Quatrani, 2000). This is because UML is a general-purpose graphical language that can be used to express object-oriented designs and also with any design methodology as well. Another reason for selecting UML is its capability to provide users with a ready-to-use and expressive visual modelling language for visualising systems, as well as providing extensibility and specialisation mechanisms to extend the core concepts. Lastly it is language independent and process independent.

3.3.1 Layered System Architectural View

Another approach to model the proposed system architectural view is through a kind of layered system architecture where it consists of two main layers, with the application systems layer on the top, and component systems at the bottom as shown in Figure 3-4. Component systems consist of three other component systems or sub-layers, they are business specific layer, middleware layer and system software layer, which are layered accordingly one after another layer. Today, most developed systems tend to model this kind of layered architectural approach. This type of layered architecture organises a software system according to layers, whereby each layer is built on top of another more general layer. It is known as a better approach to systems architecture because it is able to isolate
the functionality of the application interfaces from the business specific functionality of the system. Moreover, this approach isolates the business specific layer from the rest of the underneath layers. This kind of layered architecture helps to simplify and manage the system evolution more effectively, and thus enables significant components to be reused and more tolerant to changes. It will be further illustrated in Chapter 4.

<table>
<thead>
<tr>
<th>Application Systems</th>
</tr>
</thead>
<tbody>
<tr>
<td>Component Systems:</td>
</tr>
<tr>
<td>Business-specific Layer</td>
</tr>
<tr>
<td>Middleware Layer</td>
</tr>
<tr>
<td>System Software Layer</td>
</tr>
</tbody>
</table>

Figure 3-4: A Typical Layered System Architectural View

3.3.2 System Architectural Design

In this section, the system architecture shows the high-level or abstract design of the proposed system. With reference to Figure 3-5, this proposed system shows the 3-tier architecture, mainly it consists of several components or rather engines to demonstrate the functionalities of transferring XML data from multiple XML documents to heterogeneous relational databases, and vice versa. Each of these components layer is isolated according to their functionalities in the system. Presentation layer is responsible for displaying the application user interfaces whereby the users will interact with the system through these
interfaces. Business layer focuses on the components that are responsible to process and perform business rules of the system. These components are known as objects that represent the business, both data and behaviour. And the data access layer is responsible for the data storage and retrieval management needed by the system. This layer consists of objects that know how to communicate with the place where the data storage resides. In fact, users interact with the system through the XML-based Interfaces package. 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 insert, delete or update XML data to relational tables. The query interpreter is responsible to interpret the queries actions such as insert, select, delete and update in XQuery, perform on the XML documents. The mapping engine is used to map XML schemas or DTDs to database schemas using object-relational mapping definitions. In order to transfer data from XML documents and to any database, it is necessary to map the XML document schemas, which consists of respective DTDs and XML schemas to the database schemas. The data transfer strategy is built on top of this transformation engine. The transformation engine is used to transform the decomposed XML data in Java objects to relational 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 SELECT SQL query. Lastly, the transaction engine performs database transaction services such as commit and rollback transactions during the transfer of data to relational databases in order to maintain data integrity between XML documents and relational databases.
System Architecture View

Presentation Layer

Business Layer

Data Access Layer

TCP/IP

Heterogeneous Relational Databases

Figure 3-5: System Architecture Design
3.3.3 Object-Oriented Software Development Life Cycle

As explained earlier, object-oriented software development life cycle (SDLC), which is using use case driven approach, has been selected to use in the proposed software development life cycle here. According to the object-oriented development life cycle, it consists of three main processes; they are object-oriented analysis, object-oriented design, and object-oriented implementation. The object-oriented software development life cycle is iterative and incremental in nature as shown in Figure 3-6.
The use-case model can be implemented throughout the whole life cycle of the software development activities. In fact, the main benefit of using this SDLC is that all design decisions can be traced back to user requirements, analysis, design, and implementation and testing. Thus use-case scenarios can be used as test scenarios for testing purposes. The object-oriented system development activities include:

- Object-oriented analysis which is use case driven
- Object-oriented design
- Prototyping and component-based development
- Incremental testing

3.3.4 **Object-Oriented Analysis: Use Case Driven**

The object-oriented analysis phase of software development is concerned with determining the system requirements and identifying classes and their relationship to other classes in the problem domain (Bahrami, 1999). Most software organisations have begun to adopt use case model to build their requirement model. To start off the object-oriented development life cycle of this proposed system, the first phase is the object-oriented analysis illustrated using use-case driven approach is implemented here to capture the system’s high-level requirement model. Hence building a requirement model through use cases is a primary activity in analysis phase. A use case is a sequence of actions that an actor performs within a system to achieve a particular goal (Scott, 2001). The objective of the use case model is to document what functions the system should offer to the users (William Brown, 2002). It serves the three main purposes shown below:

- To construct the developers’ view of what the users want
• To provide a starting point for discovering the object classes

• To provide a starting point for discovering the operations for each class

With the documented use case model, the software developers can understand what the users of the system wants, and they need to understand it from a developer’s point of view. Figure 3-7 is the system’s high-level use case packages, whereby it can be divided into several packages, with each encompasses multiple child use cases. This figure models how the system works together at an abstract view. It shows that the users are directly involved only in the XML-based Interfaces use case. This use case can further be divided into various child use cases, which are included in Chapter 4. The use case of the XML-based Interfaces will interact with other instances of use cases internally, namely the Query Interpreter, Mapping, Transformation and Transfer and lastly Transaction use case models. These use case models represent the various business specific domains of the system. Examples of the scenarios when users interact with the system are to insert, update, delete, and read XML data, etc, followed by transferring these XML data from XML documents to relational databases. The next section will look into the high-level design model of this system.
Figure 3-7: XML-based Middleware Use Case Packages
3.3.5 Object-Oriented Design

After the object-oriented analysis phase, the second phase of object-oriented SDLC is the object-oriented design. It is during this time that additional objects and classes of the system are identified and defined. Many people often get confused between these two phases during software development life cycle. Even though they are totally distinct domain areas, they can be intertwined. Object-oriented development is highly incremental whereby the developers have to start with object-oriented analysis, model it, create an object-oriented design, and then perform each of these processes again and again to gradually refine and complete the software. The main idea behind object-oriented design is to design and refine the identified classes, attributes, methods, structures, and associations in iterative and incremental cycles.

Here, the object-oriented design for this proposed system is modelled using class diagrams. The UML class diagram, also referred to as object modelling, is the main static analysis diagram. These diagrams show the static structure of the model (Bahrami, 1999). Generally, the class diagram shown in the next figure is the primary means to show the structure of a system to be developed. It showed the identified classes or objects and the various relationships in which they are involved. From time to time, this class diagram will be refined and modified throughout the development life cycle until it has fully accomplished modelling the final design of the proposed system. With reference to Figure 3-8, the early identified abstract classes in the diagram are representation of the objects, their relationships, and their generic structures to show the developers for ease of understanding. This object-oriented design will be explained in more detail in Chapter 4.
Lastly, after completed the object-oriented analysis and design, the next phase in line is prototyping, which will be covered in the next section.

![Diagram of an abstract class diagram](image)

**Figure 3-8: Abstract Class Diagram**

### 3.4 System Implementation

The implementation methodology chosen is the evolutionary prototyping approach. Wherever applicable in the system, object-oriented paradigm is implemented. In object-oriented system development, prototyping is one of its vital activities following object-oriented analysis and design as mentioned above. This is because object-oriented software development encourages the software developers to view problem as a system of cooperative objects. Furthermore, it also advocates incremental development (Bahrami, 1999). Therefore, the main reason for choosing the evolutionary prototyping here is because it is able to provide a framework in which change can be easily accommodated.
right up to delivery (Ince, 1991). Moreover, evolutionary prototyping approach is chosen because it is the most extensive form of prototyping. Besides, object-oriented paradigm fits itself well into evolutionary prototyping as: the development of an initial prototype, followed by a series of iterative and incremental system enhancements, which are required to apply to the prototype in order to better meet the user requirements. Therefore, the objective of prototyping is to allow a better understanding of the system requirements. Figure 3-9 illustrates the schematic of evolutionary prototyping.

![Diagram](image)

**Figure 3-9: Schematic of evolutionary prototyping (Smith, 1991)**

According to the evolutionary prototyping approach, each stage in the implementation process can be viewed as a mini project because the prototyping process requires the whole system to be implemented in modules. Each of these modules consists of its own functionality to perform certain defined tasks for each module. Additionally, prototypes are an active model that end users can see, touch, feel and experience. Therefore, usually an approved prototype is a working equivalence to paper-based design specification, with one exception, i.e. is any error can be detected earlier. Moreover, it has been said that "a picture may be worth a thousand words, but a prototype is worth a thousand pictures" (Bahrami,
1999). Not only this is true, but a prototype enables the software developers to fully understand how easy or difficult to implement some features of the system.

Besides that, evaluation phase can be performed easily if the required supporting data as input is readily available. Also, iterative and incremental testing activities must be incorporated into the design and subsequent implementation of the system throughout its whole system life cycle. Lastly, prototyping is a useful exercise at almost any stage of the development. In fact, prototyping should be done in parallel with the preparation of the functional specification (Bahrami, 1999). As a matter of fact, prototyping will result in modifications to the specifications and also reveal additional features or problems that were not detected earlier until the prototyping phase. As a result, prototyping can increase creativity because it allows for quicker user feedback, which then leads to develop a better solution.

3.5 System Testing and Evaluation

Traditionally, testing and evaluation phases take place once the proposed system has completed its implementation phase. However, as explained earlier in Section 3.4, prototyping approach consists of iterative and incremental modules and classes testing during its implementation phase. Subsequently, the testing is performed separately for each module, unit and component of the system. In the object-oriented paradigm, it means to test every class in the system. Object-oriented software development life cycle emphasises on iterative development and continuous testing until the user is satisfied with the system. It is important to take a systematic approach and test all or most combinations and permutations of the various inputs, normal and exceptional scenarios. Like most of the testing procedures
in any organisation, during any testing phase, test data has to be collected to perform the testing as it is vital to provide input to the proposed system. Here, the testing and evaluation phase is used to test and verify the system’s functionalities and bugs. The terms testing and evaluation might seem similar, but the testing stage has limited connotation of determining the correctness of something, whereas evaluation implies determination of value, as well as correctness (Smith, 1991). Additionally, the testing and evaluation stages are required to perform on the software development tools to discover whether the tools used are compatible and effective to support the proposed solutions. Moreover, the system results also need to be tested for accuracy and consistency. During the system verification phase, the white-box testing will be conducted to test how well the implementation of the system design is. Whereas the black-box testing of the system are known as experiments to test whether or not the system’s external behaviour matches that described by the requirement specifications.
For the test model purposes, this proposed system follows the testing stages as stated by Burch (1992) in Figure 3-10. However there is a slight alteration, instead of performing each testing stage independently and at the end of the development SDLC, all testing are performed simultaneously throughout the whole system development life cycle. As a matter of fact, the testing phase cuts across the four phases of the Rational Unified Process as mentioned in the earlier section, which is during the Inception, Elaboration, Construction and Transition phases. Moreover, this is an appropriate method of testing the system because whenever there is a change or modification in the system, the changes can be repaired and adapted into the system without having to perform major changes in the system. Moreover, this approach is adapted in conjunction with the evolutionary prototyping and object-oriented software development life cycle in the implementation phase.
Besides, like most of the object-oriented architecture, the identified objects or classes are component pieces, when put to work and interact together, will accomplish their purposes to form a complete system. As such the system can be viewed as a composition of integrated objects or classes that are to be integrated one by one at a time into the whole system. During implementation phase, each defined class needs to be tested and evaluated before and after the integration process. This is to ensure that the whole system works well when all the classes are integrated together. Last but not least, after the system is completed, evaluation process will be performed at the final stage to determine how well it is designed, implemented and satisfied the objectives of this dissertation.

3.6 Selection and Familiarisation of Development Tools

Before any implementation phase takes place, one of the most important decisions to make is the selection of development tools. Efforts in learning and finding the most suitable development tools is essential before developing any system to ensure that the appropriate tools are used to develop the proposed system. In doing so, some studies have to be done on the overall software design and architecture as mentioned in the earlier section are needed in order to ensure that the selected tools are suitable to be used in this development process.
Table 3-4 explains the selections of these development tools that will be used for the prototype implementation of the proposed system.

<table>
<thead>
<tr>
<th>Features</th>
<th>Selected Development Tools</th>
</tr>
</thead>
<tbody>
<tr>
<td>Programming Language and XML Processor</td>
<td>Java 2 technology which is bundled with XML Processor APIs such as JAXP, which include classes for SAX and DOM parsing in its Java 2 platform (Java 2 Standard Edition) and Borland JBuilder 8 Enterprise Edition for Java XML APIs.</td>
</tr>
<tr>
<td>Integrated Development Environment (IDE)</td>
<td>Borland JBuilder 8 Enterprise Edition for Java because it has a comprehensive IDE which include drag-drop feature and easy-to-use Java graphical user interface components that is important to develop the graphical user interface for the proposed system. For example, Java Swing packages and Java Abstract Window Toolkit (AWT) packages.</td>
</tr>
<tr>
<td>Software Modelling Tools for analysis and design phase</td>
<td>Methodology used here is object-oriented techniques for analysis and design phases using Unified Modelling Language (UML) notation. Rational Rose Enterprise is used as the drawing tool for modelling the use case, class and sequence diagrams. Microsoft Visio for drawing architectural and deployment diagrams.</td>
</tr>
<tr>
<td>Relational database platform and database driver for connectivity</td>
<td>Relational databases selected to be evaluated are Oracle, IBM DB2 and Microsoft SQL 2000. Java Database Connectivity (JDBC) is used as the database driver to establish connection to different database management systems.</td>
</tr>
</tbody>
</table>
3.7 Conclusion

In this chapter, the research methodology defined in the earlier sections consists of set of objectives and goals that need to be satisfied at every phase of the system development life cycle (SDLC). Therefore, they are the guidelines about how this dissertation is based upon. The results derived from each phase are implemented in its consecutive phase. Conclusions that are made in the literature review have driven the conceptual analysis and design of the architecture of XML-based middleware, which in turn will create an easy and convenient implementation of the XML-based interface integrating with heterogeneous relational database types. This prototype system allows software developers and information system analysts to extract data easily from XML documents and transfer these data to heterogeneous relational databases, and vice versa, without having to acquire in-depth of XML and SQL knowledge. Finally, testing strategies are formed to test and evaluate the proposed software design plus the objectives deviation that occurred during its implementation.