CHAPTER 4 OBJECT-ORIENTED ANALYSIS AND DESIGN

This chapter will continue to focus more technical details on the object-oriented analysis and design process of the proposed system as explained briefly in the methodology framework of Chapter 3. The following sections will further refine and detail out the requirement model, which was identified through use case modelling, and show how the object-oriented analysis model and design model are derived from this requirement model.

4.1 Object-oriented Analysis

The object-oriented analysis techniques implemented here are using the emerging object-oriented technologies to construct, manage and integrate the identified objects to work together to form a complete application. In the object-oriented analysis phase, the first step in finding an appropriate solution to a given problem is to understand the whole problem domain of the system, what the system's responsibilities to the users are, and how the users will use the system. Generally, analysis is the process of extracting the needs of a system and what the system must do in order to satisfy users' requirements. Thus the main objective of the analysis is to capture a complete, unambiguous, and consistent picture of the requirements of the system and what the system must do to satisfy users' requirements (Bahrami, 1999). Nowadays, most software organisations have begun to adopt use case driven approach to build their user requirement models.
As explained earlier, the proposed system’s analysis phase was modelled using the use case driven approach. At this stage, one of the high-level use case packages that was defined in Figure 3-7 from the previous chapter, which is the XML-based Interfaces use case package, is refined and drawn more detailed to show its child use case diagrams. These use cases depict the scenarios interactions between the users and the proposed underlying system in the use case model as shown in Figure 4-1. This use case model shows that the users can be anyone who is using the system, for example, system analysts. It models the aspect of the data transfer scenarios between XML documents and relational databases, starting from where the users select the respective XML documents for XML queries manipulation such as to insert, update, delete or to select certain XML data across multiple XML documents before transferring the XML data to relational databases for storage. After the users have finished the insertion or modification of the respective XML data, they will need to select which databases to access to, in order to transfer the XML data into the relational databases. Besides that, the users can create new tables when inserting XML data to databases. In short, this use case model defines what services the system offers to its users by specifying directly how the responsibilities are allocated to the respective actors and their interactions with the instances of the use cases.
Figure 4-1: Use Case Model
4.1.1 Analysis Model

In this section, the analysis model defines how the identified analysis objects work together in a high-level view, leaving out the low-level specific details of the proposed system. This is accomplished by building several collaboration diagrams of the system that concentrate on describing what the system does instead of how it does. It is also known as robustness diagrams as explained in the book UML Explained by Kendall Scott (Scott, 2001). Throughout this chapter, the term collaboration diagram and robustness diagram are used interchangeable. To start off the analysis modelling process, all the use cases, which are drawn in Figure 4-1, will be detailed by analysing the text of each use case and thus identify the set of objects that will participate in that use case, and then classify the objects based to their behaviours and responsibilities according to the use case model. These objects are known as analysis classes in robustness analysis or collaboration diagram, which are the focus of the analysis model. The analysis model consists of three types of analysis classes that can illustrate each of the use cases and their interaction, namely boundary classes, entity classes, and control classes. With the aid of these analysis classes, each of the identified use cases can be mapped and traced to its respective collaboration diagram. Therefore, each of the collaboration diagrams consists of its own boundary classes, control classes and entity classes to show how these objects interact with each other to demonstrate the use case scenarios. In fact, they are representation of an abstraction of classes in the system's implementation and can be considered as the preliminary design.

For each use case identified, there is a collaboration or robustness diagram that illustrates on how these analysis classes interact to perform the use case functions. Each use case diagram is mapped and traced to its respective analysis classes modelled in the
collaboration diagram by a dashed line. Referring to the system's use case model in Figure 4-1, there are five main use cases identified, namely the Load XML Documents, Insert XML Data, Update XML Data, Delete XML Data, and Read XML Data. The objective of this object modelling activity is to define the possible various object types and their respective relationships, and then to identify their responsibilities. Thus, the collaboration diagrams or robustness diagrams are used here to demonstrate how these objects interact and perform their respective use case functions that are modelled in the analysis model.

4.1.2 Load XML Documents Use Case Scenario

In this section, the Load XML Documents use case is traced to its respective collaboration diagram with a dashed line as shown in Figure 4-2. From the figure shown, the boundary class is the XML-based Document Interface, which will interact with the system to allow the users to select and load the required XML documents for processing. The only control class is Validate & Display XML Documents. This control class carries out the use case specific behaviour, which in this instance, is to validate the selected XML documents to check for XML validity and well-formed format before displaying these XML documents to the users to perform simple XQuery manipulation on these XML data. It interacts with the entity class to provide a graphical XML-based document interface for the users to manipulate the XML data using XQuery, such as to insert XML data, delete XML data and so on, before transferring these XML data to relational databases for storage. The entity class in this case is the XML Documents. This entity class is an important XML input for the transferring of the XML data to the relational database.
4.1.3 Insert XML Data Use Case Scenario

With reference to Figure 4-3, it depicts that the Insert XML Data use case is used to trace to its respective collaboration diagram with a dashed line. Here, the boundary class is the Insert XML Data Interface, which will interact with the system to allow the users to insert new XML data to virtual XML structure stored in memory. There are three identified control classes, namely the Interpret Insert Query, Mapping between schemas, Transforms & Transfer; they carry out the Insert XML Data use case specific behaviour. The Interpret Insert Query control class is responsible to interpret the insert function when the user inserts new XML elements, attributes, and values into the structure of XML documents. The Mapping between schemas control class is responsible to map and serialise the XML object schemas to the database schema based on a system generated map file. It has to
interact with the XML Documents entity class during the mapping process. After the mapping has taken place, the Transforms & Transfer control class will transform these XML objects into Java objects based on the mapped schema in the first place. After that, it will transfer these persistent Java objects, which stored these XML data, to upload to the relational database. At the same time, it also handles creation of database connections from connection pooling and manages the transaction activities during the data transfer process. The entity class here is the Relational Database, which are used to store the transferred XML data into the relational table’s format. It consists of long-lived system information, which in this case is the XML data.

![Diagram](image)

**Figure 4-3: Insert XML Data Collaboration Diagram**
4.1.4 Update XML Data Use Case Scenario

Figure 4-4 depicts that the Update XML Data use case is used to trace to its respective collaboration diagram with a dashed line. The boundary class is the Update XML Data Interface. The Update XML Data use case is also one of the XML query manipulations, therefore the control classes are almost similar, except the Update XML Data Interface and Interpret Update Query. On the contrary, the similar control classes include the Mapping between schemas and Transform & Transfer; they carry out the Update XML Data use case specific behaviour. In this case, the update function is used to update the XML data over the virtual XML structure displayed before transferring to database for updating. These updated XML data, which is stored in the XML Documents entity class, will go through the Mapping between schemas control class to map the XML schema to database schema via a map file before being transformed and serialised into Java objects. Subsequently, the XML data stored in these persistent Java objects will be transferred to the relational database for storage. The entity class used here is the Relational Database, which are used to store the updated XML data to the relational tables.
4.1.5 Delete XML Data Use Case Scenario

Figure 4-5 depicts the traceability of the Delete XML Data use case to its respective collaboration diagram with a dashed line. The boundary class is the Delete XML Data Interface. Similarly, this collaboration diagram consists of three control classes, namely the Interpret Delete Query, Mapping between schemas, Transform & Transfer. They are responsible to perform the Delete XML Data use case specific behaviour. These XML data are stored in the XML Documents entity class before being deleted. In addition, the Relational Database entity class is used to update and store the deleted XML data to the relational tables.
4.1.6 Read XML Data Use Case Scenario

Figure 4-6 depicts the traceability of this use case to its respective collaboration diagram with a dashed line. The boundary class is the Database Connector Interface, where the user will interact with the back-end database systems. In order to select and read the data from the relational tables and transform into a valid XML document and its respective DTD, there are several steps involved in doing so. Firstly, the Transfer & Transform control class interacts with the Relational Database entity class to select data from a single table or multiple tables using SQL SELECT or JOIN command. After extracting these relational data, they will be transferred and downloaded to the client's side. Following that, the Transfer & Transform control class will transform and generate these XML data into valid XML documents. Next, this control class will interact with the XML Documents entity class to store the generated XML documents.
4.2 JXDB System Architecture

JXDB is a GUI-based XML middleware system designed 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 is designed to be a consistent and extensible XML middleware framework. In general, the system architecture describes the overall organisation of the system into several components known as subsystems. This architecture view provides the context in which detailed decisions are made in the later software design stages. JXDB’s architecture is organised in such layering approach whereby each layer is built on top of another more general layer and the lower layer has no knowledge of the layers above it. In fact, layered architectures are among the most frequently used high-level structures for a system. A schematic of the general layered architecture proposed for JXDB system is shown in Figure 4-7. This layered architecture
defines and constructs these static subsystems, which will integrate with each other during compilation, linking and execution time.

The top-most layer, which is known as the application system layer, contains the application system of the JXDB that provides the XML-based document interfaces to its end users. Next, below this layer is the business specific layer that consists of components systems such as the Query Interpreter, Mappings, Transformation & Transfer, and Transaction. These component systems are usually designed to be reusable and are specific to a certain type of business processing. Each of these components interacts with each other to carry out the business processing process of this system, such as to interpret XML queries over the XML documents, map XML schema to database schema, and transform to Java objects before transferring the XML data to multiple relational databases for storage and retrieval.

The second last layer is the platform independent layer. The platform independent layer offers component systems that provide utility classes and platform independent services. In this layer, these component systems used are known as middleware and they are often platform independent component systems. These interoperable component systems used here are Java 2 for Java platform independent programming language, Java DOM Parser to parse XML documents using DOM API, Java SAX Parser to parse XML documents using event-driven approach, and JDBC is Java API for connecting and accessing to the relational databases. As a result, these underlying reusable and platform independent components will enable the system to be more tolerant to the rapid technology changes especially in the ever-growing XML technologies industry. Lastly, the bottom layer, which is the system software layer, which consists of different operating systems used in this implementation
environment, different types of relational databases used, as well as TCPIP as the underlying network communication protocol used.

In an object-oriented system, the design of a system has two basic levels, namely the system design and detailed design. The system design described in this section is concerned with the overall architecture of the system and the setting of standards. On the other hand, the detailed design is concerned with designing individual components to fit this architecture and to conform to these standards, which is the design of objects. The detailed design will be described in the later sections where it is dealt with the design class diagram and sequence diagrams.
Layered System Architecture of XML-based Middleware System

Application Systems Layer

XML-based interfaces

<<invoke>>

Business Specific Layer

Query Interpreter <<using>> Mappings <<using>> Transformation & Transfer <<using>> Transaction

Platform Independent Layer

Java 2 <<using>> Java API for XML Processing <<using>> Java Architecture for XML Binding <<using>> JDBC

System Software Layer

Microsoft Windows Server 2000
Linux RedHat
Oracle Database
Microsoft SQL 2000 Database
MySQL Database
TCP/IP

Figure 4-7: Layered Architecture of the JXDB XML-based System
4.3 Object-oriented Design

After performing the object-oriented analysis activities illustrated using use case models and analysis models, the next step in line is to start off the object-oriented design activities. Here, the object-oriented design centers on establishing the design classes, attributes, methods and their protocols by building class diagrams. Similarly, the use cases explained in the previous section can be employed throughout the design activities. Besides that, the object-oriented design is also iterative and incremental in nature. During the object-oriented design phase, the design model is elevated into actual objects that perform the required task as defined from the previous analysis phase. The goal of object-oriented design is to design the classes identified during the analysis phase and the user interface (Bahrami, 1999). In fact, there is an emphasis shift from the application domain to the implementation phase. Therefore, the design model specifies the high-level view of solving the given problem domain and thus constructs an appropriate solution to it. In short, it consists of the detailed design classes and objects, and design subsystems that correspond more closely to the final system implementation phase.

4.3.1 Design Model

During the design model phase, critical decisions are made on how to solve the problem domain of this proposed XML-based middleware and how to implement the solved problems into the proposed system. As described before, it starts off by solving the problem domain from the top-most level, then iteratively and increasingly towards the detailed levels. The design model is organised into several subsystems, which consists of lower level of subsystems and objects or classes from the analysis phase. The objects discovered
during the previous analysis phase can serve as the framework for the design phase. As a result, during the transition phase from analysis classes to design classes, more details related to the implementation phase are revealed and implemented. Object-oriented design process requires taking the objects identified during the object-oriented analysis and designing classes to represent them. A design class represents a class at a more detailed, but still at high level in the system's implementation. Additionally, when designing and refining the design classes, some new design classes might be added, removed, spitted or even combined. Furthermore, there should be obvious traceability and mappings from the use case models, to their respective analysis models and design models, whose analysis models provide the foundation for the design models. In this case, the design model consists of a set of sequence diagrams and a class diagram.

4.3.2 Sequence Diagrams

Before proceeding to build the design classes and their relationships, it is best to use the interaction diagrams to show the detailed message flows as well as the underlying objects' interaction details. Interaction diagrams are used to capture the behaviour of a single use case and also to show the pattern of interaction among objects by describing how these groups of objects collaborate to perform their assigned jobs. Sequence diagram is one of the interaction diagrams used to model the dynamic behaviour of the system's use case instances. The UML sequence diagram is designed to focus on the time ordering of the messages that go back and forth between the objects. Once the sequence diagram is drawn, information flows and their sequences among the identified objects defined earlier in the analysis model can be seen more clearly. Here, this diagram reflects the dynamic behaviour
of the objects and their activities during their interactions with each other when performing their use case behaviour instances. In fact, when a sequence diagram is used to model the dynamic behaviour of a use case, it can be seen as a detailed specification of the use case (Bennett et al., 2002). Moreover, sequence diagrams belong to the design model that holds the design diagrams. Most importantly, these sequence diagrams are used to decide where to assign the classes' operations, based on the methods that they assign to objects on the sequence diagrams. Each use case is traced from the robustness or collaboration diagrams, which is the foundation for the design model, and the details of the operations can be illustrated by using these sequence diagrams. The use case instances that depict the use case model, which are defined in Figure 4-1, are now mapped and traced through to their respective activities using sequence diagrams. These sequence diagrams depict the object interactions from their collaboration diagrams by their lifelines and the messages exchange with each other arranged in a time sequence. The focus of control shown in the sequence diagram indicates times during an activation when processing is taking place within that object (Bennett et al., 2002).

4.3.2.1 Load XML Documents Sequence Diagram

Figure 4-8 shows the sequence diagram for Load XML Documents, which is mapped and traced from its respective collaboration diagram shown in Figure 4-2. The vertical dimension represents time and all objects involved in the interaction are spread horizontally across the diagram. Each of these objects is represented by a vertical dashed line, known as lifeline, with an object symbol at the top, and in this case, the objects are the actors and XMLInterfaces, the latter is the boundary object identified from the collaboration diagram
earlier, but here it is shown as one of the objects in this sequence diagram. The Validate & Display XML Documents control object receives a message from XMLInterfaces object to validate selected XML documents and at the end of activation it will display the XML views by sending a return message back to XMLInterfaces object. A message is shown by a solid horizontal arrow from one lifeline to another and is labelled with the message name.

Figure 4-8: Load XML Documents Sequence Diagram

4.3.2.2 Insert XML Data Sequence Diagram

Figure 4-9 shows a straightforward sequence diagram drawn from its respective collaboration diagram shown in Figure 4-3. Here, the boundary object is shown as the interface class that the actor interacts with when using the system via the InsertXMLInterface class, whereas the control objects are drawn as the rest of the participating classes. The sequence diagram starts to model when the actor selects and loads the required XML documents to work with. Next, the actor begins inserting any new element or attribute data into the loaded virtual XML document, and this will invoke the
4.3.2.3 Update XML Data Sequence Diagram

Similarly, Figure 4-10 shows a straightforward sequence diagram drawn from its respective collaboration diagram shown in Figure 4-4. The boundary object is modelled as the UpdateXMLInterface class. The participating objects or classes shown in this sequence diagram are the control objects taken from its collaboration diagram. To elaborate this sequence diagram, the flows of events begin with the actor updating the XML data and this XML update query is processed by the UpdateQueryInterpreter class. Following that, the UpdateXMLInterface class calls the Mapping class or object to perform the mapping of XML schema to database schema. After receiving an acknowledgement back from the Mapping class, the UpdateXMLInterface class will summon the Transformation class to perform the necessary transformation of XML data to Java objects. Subsequently, the
Transfer class will transfer and upload these data stored in Java objects to the relational database. Meanwhile, the DBTransaction class is responsible to select the required database and establish the connection by sending a call to the Relational Database class. When the DBTransaction class has received an acknowledgement from the Relational Database class, it will initiate the transaction activities, whereas the latter class will start to execute SQL UPDATE statement and commit the transaction if there were no data failure. Upon completion of updating data to the database, the DBTransaction class will send an acknowledgement back to the actor to end the whole process.

![Update XML Data Sequence Diagram](image)

**Figure 4-10: Update XML Data Sequence Diagram**
4.3.2.4 Delete XML Data Sequence Diagram

Figure 4-11 depicts almost similar pattern of interaction as the earlier sequence diagrams shown in Figure 4-9 and Figure 4-10. Here, the boundary object is modelled as the DeleteXMLInterface class and the control objects are also drawn as the participating classes in this diagram. It starts with the actor loads the XML document and deletes XML data in this XML document, and this will invoke the DeleteQueryInterpreter class to interpret and process the XML delete query. After the DeleteQueryInterpreter has finished processing the delete action, the DeleteXMLInterface class will invoke the Mapping class to do the mapping from XML schema to database schema based on a predefined built-in mapping file. After the DeleteXMLInterface class has received an acknowledgement from the Mapping class, it will invoke the Transformation class. Here, the Transformation class will perform the transformation of XML data to Java objects. Following that, this class will summon the Transfer class to transfer and upload the modified XML data to the database. Prior to that, the DBTransaction class will establish the database connection by sending a call to the Relational Database class. When the DBTransaction class has received this acknowledgement from the latter class, it will initiate the transaction by sending a request to the Relational Database class to start executing SQL DELETE statement and commit the transaction if there were no data failure. Upon completion of deleting data from the database, the DBTransaction class will send an acknowledgement back to the actor to end this entire process.
Figure 4-11: Delete XML Data Sequence Diagram

4.3.2.5 Read XML Data Sequence Diagram

Figure 4-12 shows a straightforward sequence diagram drawn from its respective collaboration diagram shown in Figure 4-6. In this scenario is where the actor selects and reads the data returned from the relational database in XML format. The participating objects or classes shown in this sequence diagram are the control objects taken from its respective collaboration diagram. To elaborate this diagram, the flows of events begin with the actor selecting the required database to establish the database connection. After that, the Relational Database class will execute the SQL SELECT statement to select the required data from the respective tables. Next, the returned data has to be transferred and
downloaded to the client's side as sets of disconnected Java data objects, and this is done by the Transfer class. Since the query results need to be in XML format, then these disconnected Java data objects will be transformed into XML format by the Transformation class. Note that even though now almost all major relational databases can return their query results in XML format but this proposed system does not use these APIs. This is because by using these specific APIs from these database vendors will only tight this system to a specific database product. Following that, the Transformation class will transform the query results into XML format, so that the XML Interfaces class can generate a valid and well-formed XML file from these returned results. Finally, the latter class will display the formatted XML data for the actor to view and manipulate.

Figure 4-12: Read XML Data Sequence Diagram
4.4 Class Diagram

At this stage, the object-oriented design process proceeds with the next step in the development by designing a class diagram for this proposed system, which is to model the static design view of the system. According to its general definition, a class diagram is a diagram that shows a set of classes, their structures and the various relationships in which they are involved (Scott, 2001). Thus class diagrams are the primary means to show the structure of a system being developed. However, before proceeding to the system’s design classes, there should be an obvious link from the interaction diagrams, which are the sequence diagrams described above, to this class diagram. In fact, there should be full consistency between a class diagram and a set of related interaction diagrams. This is due to the fact that the preparation of interaction diagrams, either sequence or collaboration diagrams, involves the allocation of operations to classes. These operations should be listed against the correct classes in the class diagram and if the operation signatures have been specified in full these must be consistent (Bennett et al., 2002). As a result, the interaction diagrams and the class diagrams should be mutually consistent.

Figure 4-13 presents the proposed system’s class diagram. It illustrates how the objects or classes identified in the collaboration or robustness diagrams and the sequence diagrams are summarised and drawn as the design classes. At this stage, almost all the design classes shown below are without their attributes, except the XMLInterfaces class because its attributes are important to define the layout of the system’s user interface. The operations of each class are derived from the message calls that occurred during the objects’ interaction when carrying out the behaviour of the use case instances. Almost all the classes shown in this diagram are derived from the control classes of the robustness diagrams
described above, except only one that has been taken from the boundary class category, which is the XMLInterfaces. It has been drawn as one of the design classes that will handle and manage the system interaction with the actors. This class, in turn, depends on the QueryInterpreter class to interpret the actor’s XML query requests. In order to get the job done, this class has to use the services provided by some other classes and these classes, in turn, also need to use the services provided by other classes, so that they can complete the job. This is because each of these classes has been assigned specific tasks according to their roles in the system. Therefore, here the classes are shown with links connecting to related classes together if they have a relationship and interaction between them.

In the first place, the XMLInterfaces class will receive the actor’s XML query request through its user interface, and it will invoke a call to the QueryInterpreter class to process and interpret what kind of XML query request it is. This kind of relationship is dependency and it is indicated with a dotted line from XMLInterfaces to QueryInterpreter. Once the QueryInterpreter has done its task, it will need to use the Mapping class’s services to map the XML schema to the database schema, so there is a link from the QueryInterpreter class to the Mapping class to indicate their stereotype relationship together. Following that, the mapped XML schema will be used by the Transformation class to perform the required transformation on the XML data, which is to transform these XML data to Java data access objects before sending them to Transfer class for further processing. Again, there is a line linking the Mapping class and the Transformation class together as both of them shared a stereotype of using relationship. Now, once the Transfer class is activated by the Transformation class, it is responsible to transfer these Java data access objects to the DBTransaction class for database transaction and processing activities. This stereotype using relationship, which connects the Transformation class and the Transfer class together,
is also indicated by a line linking them because the Transfer class is using the services provided by the Transformation class. Similarly, the Transfer class is connected to the DBTransaction class via the stereotype using relationship. In this case, the DBTransaction class is responsible to manage the database connections, transactions, and database transaction activities. Lastly, the Database class is responsible for inserting, updating, deleting, selecting, creating or dropping of relational tables in the respective database. It is associated with the DBTransaction class and Transfer class.

![Class Diagram](image)

Figure 4-13: Class Diagram
4.5 Summary

In summary, this chapter concentrates on the important steps of the object-oriented analysis and design processes, which are needed to design quality software architecture, in terms of reusability, maintainability, and robustness. Object-oriented analysis provides a detailed discussion of use case driven approach and how to develop these use cases. The main goal of the analysis process is to capture a complete, unambiguous, and consistent picture of the system's requirements and what the system must do to satisfy the users' requirements by developing the use case model of the system. This is accomplished by constructing several collaboration diagrams of the system that can be traced from their respective use case diagrams. These diagrams concentrate on describing what the system's specific use case behaviour do rather than how the use cases do it. In contrast, the latter process concentrates on the basic steps undertaken in the object-oriented design process, like modelling the system's use case dynamic behaviour using sequence diagrams to determine the design classes' methods, which can be mapped and traced from its respective collaboration or robustness diagrams, then applying the identified design classes, their attributes, methods, associations, structures, and protocols. These design classes are continuously refining and detailed out in an iterative process until they fully represent the actual software design architecture. The single most important activity in designing an application is to develop a set of classes that can interact together to provide the required system functionalities. After all, underlying the functionalities of any application is the quality of its software design.