

**SYNCHRONIZATION OF REMOTE DATABASES**

**THESIS**

Presented to the Graduate Council of Texas State  
University-San Marcos in Partial Fulfillment of  
the Requirements

For the Degree

Master of SCIENCE

By

Jasmine Pabby

San Marcos, Texas  
December 2003

**COPYRIGHT**

**By**

**Jasmine Pabby**

**2003**

## ACKNOWLEDGEMENTS

Completing this thesis with my very challenging instructor Dr. Haddix gives me special pride that I survived it all. Thank you Dr. Haddix, I have learnt extensively from you and at the end of it all find myself so much more knowledgeable and confident. You have taught me how to be uncompromising towards quality. Thank you also for your immense support and patience which have meant a lot to me.

I have also come to understand the meaning of teamwork with more enforced significance. I cannot thank Dr. Ali enough, whose immense support in every way made it possible for me to believe in myself and to continue working hard. Thank you Dr. Hazlewood and Dr. Ogden, who have been very accommodating with the last minute schedules. They are two of my most favorite professors and I enjoyed taking classes with them.

My brother, my son, my husband, my parents and my in-laws--who would not let me quit and tolerated my different moods this semester--I cannot thank them enough. I also owe a special thanks to the Graduate College, especially Heather, who besides having a very pleasing personality, has been so very prompt and helpful with information.

## TABLE OF CONTENTS

<b>ACKNOWLEDGEMENTS .....</b>	<b>IV</b>
<b>LIST OF TABLES .....</b>	<b>VIII</b>
<b>LIST OF FIGURES .....</b>	<b>IX</b>
<b>ABSTRACT .....</b>	<b>X</b>
<b>CHAPTER 1: INTRODUCTION .....</b>	<b>1</b>
<b>1.1 Introduction to the Problem.....</b>	<b>1</b>
<b>1.2 Identifying the Problem .....</b>	<b>2</b>
<b>1.3 The Objective.....</b>	<b>3</b>
<b>1.4 The ‘Unit Order of Battle’ Database .....</b>	<b>4</b>
<b>1.5 The ‘Functional Descriptions of the Mission Space’ Database .....</b>	<b>5</b>
<b>1.6 Remaining Chapters .....</b>	<b>6</b>
<b>CHAPTER 2: REQUIREMENTS SPECIFICATION .....</b>	<b>8</b>
<b>2.1 Restrictions in Systems Specification .....</b>	<b>8</b>
<b>2.2 Mapping UOB Tables to FDMS Tables .....</b>	<b>8</b>
2.2.1 Mapping UOB Tables to the Entity Table in the FDMS Database.....	9
2.2.2 Mapping UOB Tables to the Entity-Component Tables in the FDMS.....	14
2.2.3 Mapping UOB Attributes to Characteristic Table in the FDMS Database.....	17
2.2.4 Mapping UOB Tables to Entity-Characteristic Table in the FDMS.....	18
2.2.5 Tracing and Updating the Database.....	24
<b>2.3 Systems Specification .....</b>	<b>25</b>
2.3.1 Transmission side (UOB).....	25
2.3.2 Receiving Side (FDMS) .....	28
<b>2.4 Stages of Operation .....</b>	<b>30</b>
<b>2.5 Summary of the Chapter .....</b>	<b>37</b>
<b>CHAPTER 3: ENABLING TECHNOLOGIES AND OUR SYSTEM. 38</b>	
<b>3.1 Choosing the Enabling Technologies.....</b>	<b>38</b>

<b>3.2</b>	<b>Choice of Distributed Enterprise Technology .....</b>	<b>38</b>
3.2.1	XML Portal Server (XPS) .....	39
3.2.2	.Net .....	40
3.2.3	Java 2 Enterprise Edition (J2EE).....	42
<b>3.3</b>	<b>Other Supporting Technologies .....</b>	<b>43</b>
3.3.1	Operating System and J2EE .....	43
3.3.2	Programming Language and J2EE .....	43
3.3.3	Database Management System and J2EE.....	43
<b>3.4</b>	<b>Java 2 Enterprise Edition (J2EE) and Our System.....</b>	<b>43</b>
3.4.1	Presentation Tier.....	45
3.4.2	Web Tier.....	45
3.4.3	Application Tier .....	45
3.4.4	Database Tier.....	46
<b>3.5</b>	<b>J2EE Components and Our System .....</b>	<b>46</b>
3.5.1	Web Components and Our System.....	48
3.5.2	Application-Server Components and Our System.....	49
<b>3.6</b>	<b>J2EE Containers and Our System .....</b>	<b>50</b>
3.6.1	Web Contamer and Our System .....	50
3.6.2	EJB Container and Our System.....	51
<b>3.7</b>	<b>J2EE and Communication.....</b>	<b>51</b>
3.7.1	Communication Services.....	53
3.7.2	Enterprise Services .....	54
3.7.3	Internet Services .....	55
<b>3.8</b>	<b>Installed Software.....</b>	<b>55</b>
<b>3.9</b>	<b>Summary of the Chapter .....</b>	<b>58</b>
<b>CHAPTER 4: SYSTEM DESIGN.....</b>		<b>59</b>
<b>4.1</b>	<b>Introduction to the System Design.....</b>	<b>59</b>
<b>4.2</b>	<b>Design of the UOB System.....</b>	<b>60</b>
4.2.1	Design of the Presentation Tier.....	62
4.2.2	Design of the Web Tier .....	65
4.2.3	Design of the Application Tier .....	69
4.2.4	Design of the Database Tier .....	71
<b>4.3</b>	<b>Design of the Receiving Side (FDMS) System.....</b>	<b>71</b>
4.3.1	Design of the Database Visualizer.....	72
4.3.2	Design of the Application Tier .....	73
4.3.3	Design of the Database Tier .....	75
<b>4.4</b>	<b>The Complete System Architecture.....</b>	<b>76</b>
<b>4.5</b>	<b>Summary of the Chapter .....</b>	<b>77</b>
<b>CHAPTER 5: PROJECT IMPLEMENTATION .....</b>		<b>78</b>

<b>5.1</b>	<b>Introduction to implementation .....</b>	<b>78</b>
<b>5.2</b>	<b>Transmission Side .....</b>	<b>78</b>
5.2.1	Implementation Description of Transmission Side Modules.....	78
5.2.2	Component Modules of the Transmission Side (UOB).....	100
<b>5.3</b>	<b>Receiving Side .....</b>	<b>112</b>
5.3.1	Implementation Description of Receiving Side Modules.....	112
5.3.2	Component Modules of the Receiving Side (FDMS).....	122
<b>5.4</b>	<b>Summary.....</b>	<b>129</b>
 <b>CHAPTER 6: CONCLUSIONS AND FUTURE WORK.....</b>		<b>130</b>
<b>6.1</b>	<b>Conclusions .....</b>	<b>130</b>
<b>6.2</b>	<b>Future Work .....</b>	<b>131</b>
 <b>GLOSSARY .....</b>		<b>132</b>
 <b>APPENDIX A: UOB DATABASE TABLES .....</b>		<b>138</b>
 <b>APPENDIX B: DMS DATABASE TABLES .....</b>		<b>141</b>
 <b>APPENDIX C: CODE .....</b>		<b>146</b>
 <b>REFERENCES.....</b>		<b>258</b>
 <b>BIBLIOGRAPHY .....</b>		<b>260</b>

## LIST OF TABLES

TABLE 2.1: MAPPING UOB TABLES TO FDMS ENTITY TABLE .....	13
TABLE 2.2: MAPPING UOB TABLES TO FDMS ENTITY-COMPONENT TABLE .....	16
TABLE 2.3: MAPPING UOB TABLES TO FDMS CHARACTERISTIC TABLE .....	18
TABLE 2.4: MAPPING UOB TABLE TO FDMS ENTITY-CHARACTERISTIC TABLE.....	22
TABLE 2.5: MAPPING UOB RECORDS TO MULTIPLE ENTRIES IN FDMS .....	23
TABLE 4.1: TABULAR REPRESENTATION OF WEB TIER DESIGN (CONTINUED IN 4.1(A))....	67
TABLE 4.1 (A): TABULAR REPRESENTATION OF WEB TIER DESIGN.....	68
TABLE 4.2: TABULAR REPRESENTATION OF APPLICATION TIER ACTIVITIES .....	70
TABLE 5.1: USER LOGIN (CONTINUED ON 5.1(A)) .....	80
TABLE 5.1 (A): USER LOGIN .....	81
TABLE 5.2: UOB DATABASE TABLES (CONTINUED IN TABLE 5.2 (A)).....	83
TABLE 5.2(A): UOB DATABASE TABLES (CONTINUED IN TABLE 5.2 (B)).....	84
TABLE 5.2 (B): UOB DATABASE TABLES (CONTINUED IN TABLE 5.2 (C)) .....	85
TABLE 5.2 (C): UOB DATABASE TABLES .....	86
TABLE 5.3: ACCESSING TABLE TO CREATE ROW .....	89
TABLE 5.4: CREATING A NEW ROW .....	90
TABLE 5.4: DELETING A ROW .....	93
TABLE 5.5: ACCESSING THE TABLE TO EDIT DATA.....	95
TABLE 5.6: EDITING THE DATA IN THE TABLE.....	98
TABLE 5.7 CREATING A NEW ROW IN DATABASE.....	114
TABLE 5.8: DELETING A ROW IN DATABASE .....	117
TABLE 5.9: UPDATING (EDIT-DATA) THE DATABASE.....	119

## LIST OF FIGURES

FIGURE 1.1 UNIT ORDER OF BATTLE (UOB) DATABASE.....	4
FIGURE 1.2: ABBREVIATED FUNCTIONAL DESCRIPTION OF THE MISSION SPACE (FDMS) DATABASE .....	6
FIGURE 2.1: REQUIREMENTS SPECIFICATION -UOB SIDE .....	27
FIGURE 2.2: REQUIREMENTS SPECIFICATION – FDMS SIDE.....	29
FIGURE 2.3: LOGIN PAGE.....	30
FIGURE 2.4: DATABASE ACCESS.....	30
FIGURE 2.5: EDITING A UOB TABLE .....	31
FIGURE 2.6: PROCESSING THE DATA.....	31
FIGURE 2.7: UPDATING UOB DATABASE .....	32
FIGURE 2.8: UPDATED DATABASE DISPLAYED .....	32
FIGURE 2.9: DATA MAPPING.....	33
FIGURE 2.10: DATA TRANSFERRED FROM UOB DATABASE .....	33
FIGURE 2.11 DATA TRANSFER TO FDMS DATABASE.....	34
FIGURE 2.12 FDMS DATABASE UPDATED .....	34
FIGURE 2.13 FDMS TABLES DISPLAYED .....	35
FIGURE 2.14 STAGES OF OPERATION .....	36
FIGURE 3.1: J2EE ARCHITECTURE.....	44
FIGURE 3.2: J2EE COMPONENTS .....	47
FIGURE 3.3: J2EE CONTAINERS.....	51
FIGURE 3.4: J2EE STANDARD PROTOCOLS.....	52
FIGURE 3.5: PUBLISH-SUBSCRIBE MESSAGING MODEL.....	54
FIGURE 3.6: INSTALLED SOFTWARE.....	57
FIGURE 4.1: THE SYSTEM DESIGN .....	60
FIGURE 4.2: THE TRANSMISSION SIDE (UOB) DESIGN.....	62
FIGURE 4.3: PICTORIAL REPRESENTATION OF PRESENTATION TIER DESIGN.....	64
FIGURE 4.4: TRANSMISSION SIDE (FDMS) DESIGN .....	72
FIGURE 4.5: TRANSMISSION SIDE (FDMS) DESIGN .....	75
FIGURE 4.6: SYSTEMS ARCHITECTURE.....	76
FIGURE 5.1: USER LOGIN .....	79
FIGURE 5.2: LOGIN ERROR .....	82
FIGURE 5.4: CREATING A NEW UNIT AIRCRAFT TABLE.....	88
FIGURE 5.6: EDITING THE TABLE .....	97
FIGURE 5.7: DATA ACCEPTED BY RECEIVING SIDE (FDMS) .....	112
FIGURE 5.8: DATA TRANSFERRED TO APPLICATION SERVER (FDMS).....	113
FIGURE 5.9: FDMS DATA DISPLAY .....	121

## **ABSTRACT**

### **SYNCHRONIZATION OF REMOTE DATABASES**

Most modern systems work in networked environments. Data used by these programs is distributed and stored on heterogeneous systems, in multiple databases. Synchronizing this data across multiple systems or providing an integrated access to it is a challenging issue. The Department of Defense (DoD) has two such databases, The Unit Order of Battle (UOB) and the Functional Descriptions of the Mission Space (FDMS). In this dissertation, we will demonstrate the synchronization of the data between the two databases, using standard, open-source J2EE technology and communicating through the Internet using common, open communication protocols.

by

JASMINE PABBY, M.S.

Texas State University-San Marcos

December 2003

SUPERVISING PROFESSOR: DR. FURMAN HADDIX

# CHAPTER 1

## INTRODUCTION

### 1.1 Introduction to the Problem

Most modern systems work in networked environments. The systems are distributed across wide area networks, and geographical locations hold few restrictions. Data used by these programs are thus distributed and stored on disparate systems. The management and synchronization of this data, spread out over the network, becomes an active problem.

Providing an integrated access to multiple heterogeneous sources is a challenging issue in global information systems for cooperation and interoperability. In the past, companies have equipped themselves with data storing systems building up informative systems containing data that are related one another, but which are often redundant, heterogeneous and not always substantial. The problems that have to be faced in this field are mainly due to both structural and application heterogeneity, as well as to the lack of a common ontology, causing semantic differences between information sources.

[BDBS00].

This topic has been an active study of research, although most of it has concentrated on the problem of dynamic data integration. “The standard approach to this problem has been to construct a global schema that relates all the information in the different sources

and to have the user pose queries against this global schema or various views of it. The problem with this approach is that integrating the schemas is typically very difficult, and any changes to existing data sources or the addition of new ones requires a substantial, if not complete, repetition of the schema integration process” [YA96].

Dr Furman Haddix and Jack Sheehan in their paper have described how data is transferred utilizing XML-based using Conceptual Models of the Mission Space DIF [SJHF00].

A slightly different approach has been the use of “Mobile Agents” along with the use of mediators, which react to changes, travel to the areas of change, carrying with them the agents for managing the ‘change’ [BDBS00].

We have used a different approach in addressing this issue. Our focus is to synchronize remote, disparate databases. We will solve this using off-the-shelf, asynchronous communication technology, communicating over the Internet using commonly used protocols. We will not be using mediators. Instead, each database will communicate to a common pool as a publisher of change. The interested database will listen to this pool, pick up the change and possess the local intelligence to use the information for synchronization of its data.

## **1.2 Identifying the Problem**

The Department of Defense (DoD) has several databases, situated on different systems, at different locations. Some of these databases, although disparate in nature, contain segments of common information. When data in one database is modified, data fields that are common to fields in other databases are no longer consistent. Decisions made on

inconsistent data could result in significant mistakes. Hence, there exists a need to create a methodology to synchronize common data fields in disparate databases.

### **1.3 The Objective**

Our objective is to create an electronic system that would synchronize databases existing at different locations. We will assume, while scoping the design and architecture of the system, that the computers housing the databases we are required to synchronize will be connected to a network and will have the basic ability to communicate with each other through the network via a common, standard communication protocol. We have selected two as the focus of this work: the Unit Order of Battle (UOB) and the Functional Descriptions of the Mission Space (FDMS). Both of these databases belong to the Department of Defense and reside on systems in geographically different locations.

## 1.4 The ‘Unit Order of Battle’ Database

The Unit Order of Battle (UOB) database collects mass data about operational units. It contains information about military organizations for simulation knowledge acquisition, scenario development, and operational planning. UOB is defined as data about military organization units, their organizational relationships, and their associated personnel, aircraft, and equipment [HSH99]. Figure 1.1 illustrates a version of the UOB database simplified by excluding attributes not relevant to this thesis. This database contains data in four related tables [UOB99]:

- ◆ Unit – e.g., id, name, location, type, parent.
- ◆ Personnel – e.g., grade occupation, quantities.
- ◆ Equipment – e.g., Type, quantities.
- ◆ Aircraft – e.g., descriptions, quantities.

### Unit

Unit-Identification-Code
Parent-Unit-Identification-Code
Unit-Name
Home-Name
Ship-Category
Country-Code

### Unit Equipment

EIC
Unit-Identification-Code
Equipment-Code
Equipment-Description
Equipment-Qty-Required
Equipment-Qty-Authorized

### Unit Personnel

PIC
Unit-Identification-Code
Personnel-Description
Personnel-Qty-Required
Personnel-Qty-Authorized

### Unit Aircraft

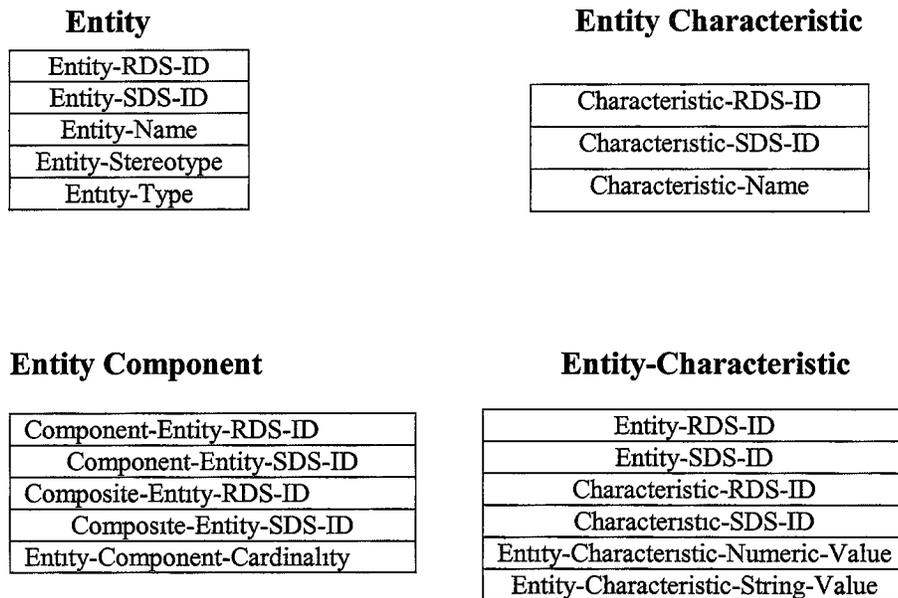
AIC
Unit-Identification-Code
Aircraft-Code
Aircraft-Description
Aircraft-Qty-Required
Aircraft-Qty-Authorized

**Figure 1.1: Unit Order of Battle (UOB) Database**

### **1.5 The 'Functional Descriptions of the Mission Space' Database**

The Functional Descriptions of the Mission Space (FDMS) has been developed for the exchange of functional descriptions of military operations without loss or distortion of content. It was created to put the requirements description process under the control of the warfighter, so that the ultimate expression of the functional requirements for a system was appropriate for warfighter intent [HaFu01]. FDMS therefore contains specific data about military operations and is used as an aid for the development of simulations. A fundamental objective of FDMS is to provide simulation/federation developers with timely and cost-effective access to accurate mission space functional models that are created, authenticated, and maintained by others [FDMS00].

Some of the data used by FDMS is originally collected and stored in the UOB database. When changes occur in data fields of the UOB database, we will update the corresponding fields of the FDMS database. The mapping between the two databases is discussed in the next chapter. Figure 1.2 illustrates the part of the FDMS database where the UOB data to be updated resides.



**Figure 1.2: Abbreviated Functional Description of the Mission Space (FDMS) Database**

## 1.6 Remaining Chapters

In **Chapter 1** we identified our problem, defined the objective and introduced the two databases that we will synchronize. We also specified fields we will focus on, to synchronize the data.

In **Chapter 2** we discuss systems requirements and the step-by-step stages of operation that will be taken in the process of database synchronization.

**Chapter 3** explores the different media available for the requirements design and implementation and explains our selection of the software found most appropriate for the final solution specifically J2EE. It also defines the enabling technology (J2EE) and its components, and describes how the technology helps build our system.

**Chapter 4** explains and illustrates our system design and its architecture.

**Chapter 5** describes the high-level and the low-level user and system implementation process of database synchronization.

**Chapter 6** draws conclusions to our system design and functions and suggests future work to make it more efficient

## **CHAPTER 2**

### **REQUIREMENTS SPECIFICATION**

#### **2.1 Restrictions in Systems Specification**

There were restrictions on the availability of government data encountered while developing the system specification:

- ◆ It was found that the UOB and the FDMS databases would not be physically available to us. For this reason, data was created in two representative databases (see Appendix A for example data). The databases were modeled after the actual UOB and FDMS databases, respectively, to be representative of each of them. In addition, column contents had to be mapped between the two databases to allow synchronization as described in the next section.
- ◆ A data-entry user interface will be created (presentation tier) to allow modifications to the representative UOB databases.
- ◆ A Database Visualizer application will be installed to view the contents of the representative FDMS database – before and after data is synchronized in it.

#### **2.2 Mapping UOB Tables to FDMS Tables**

The data in the UOB tables has multiple records in the FDMS database. Tables 2.1 through Table 2.4 display the mapping of UOB tables to the FDMS tables. Section 2.2.5

displays the multiple entries recorded in the FDMS tables for every column from the UOB tables. A complete mapping description of how the data will be updated is described in section 2.2.6.

### **2.2.1 Mapping UOB Tables to the Entity Table in the FDMS Database**

Table 2.1 illustrates which columns in the UOB tables map to the various columns of the Entity table in the FDMS database. The Entity table contains basic information about things. This information from all UOB tables is recorded into it.

Mapping description from each UOB table to the Entity table is as follows:

➤ **Unit Table:**

- ◆ ‘Unit Identification Code’ (UIC) in the Unit table, in the UOB database, is a very important column. It is a foreign key in the other tables, relating data from the other UOB tables back to the Unit, described in the Unit table. The ‘UIC’ is recorded in the personnel, equipment and aircraft tables in the UOB database. Each row in these tables can be uniquely identified as a combination of the identification code of that table (for example, ‘Personnel Identification Code’ (PIC) in the Unit Personnel table) with the ‘UIC’ of the Unit table (UOB tables 2, 3 and 4 in Appendix A). The values in the ‘UIC’ are recorded in the ‘Entity SDS ID’ column of the Entity table, in the FDMS database.
- ◆ The ‘Unit Name’ column in the Unit table, in the UOB database, describes the category the ‘UIC’ belongs to. The values in the ‘Unit Name’ column are recorded in the ‘Entity Name’ column in the entity table.
- ◆ The values in the ‘Home Name’ and ‘Ship Category’ columns are local to the UOB tables and are not recorded in the FDMS database.

- ◆ The column 'Entity Stereotype' contains the value 'Organization' and the column 'Entity Type' contains the value 'Unit' identifying that the information inserted in the row is for the Unit table. The 'Entity Stereotype' and 'Entity Type' columns are local to the 'Entity' table in the FDMS database.
  - ◆ A surrogate key is placed in the 'Entity RDS ID' column of the Entity table to uniquely identify the rows of information in the table. This key is used to relate rows in the Entity table to the rows in the Entity Component and Entity Characteristic tables. For example, if there is an 'Entity RDS ID' of '103', it has the UIC (UOB) of value 'ARMR CO', has the 'Entity Name' 'Armor Company', is of the 'Stereotype' 'Organization' and belongs to the 'Entity Type' 'Unit'.
- Personnel Table:
- ◆ PIC (Personnel Identification Code) in the Personnel table (UOB) is used to uniquely identify rows when combined with the values in 'UIC' (see Personnel Table, UOB, Appendix A). It is recorded in the Entity table (FDMS) in the 'Entity SDS ID' column.
  - ◆ 'Personnel-Description' column in the Personnel table gives a job description of the personnel. The values in the 'Personnel-Description' column are recorded in the 'Entity Name' column in the Entity table.
  - ◆ The column 'Entity Stereotype' contains the value 'Person' and the column 'Entity Type' contains the value 'Personnel' identifying that the information inserted in the row is for the Personnel Table. 'Entity Stereotype' and 'Entity Type' are local to the Entity table in the FDMS database.

- ◆ A surrogate key is placed in the 'Entity RDS ID' column in the Entity table. The key uniquely identifies the entity name, the stereotype and the Unit the personnel belong to. Therefore the key number, '401' corresponds to PIC with value, 'LT COL', has the 'Entity Name' 'Lieutenant Colonel', is of Stereotype, 'Person', and belongs to 'Entity Type', 'Personnel'.
- Equipment Table:
  - ◆ EIC (Equipment Identification Code) in the Equipment table (UOB) can be identified uniquely when combined with the values in 'UIC' (see Equipment table, UOB, Appendix A). EIC is recorded in the Entity table (FDMS) in the 'Entity SDS ID' column.
  - ◆ 'Equipment Description' column in the Equipment table gives a brief description of the equipment. The values in the 'Equipment Description' column are recorded in the 'Entity Name' column in the Entity table.
  - ◆ The column 'Entity Stereotype' contains the value 'Equipment' and the column 'Entity Type' contains the value 'Equipment' identifying that the information inserted in the row is for the Equipment Table. 'Entity Stereotype' and 'Entity Type' are local to the Entity table in the FDMS database.
  - ◆ The surrogate key in this table uniquely identifies the entity name of the equipment, the stereotype and the Unit the equipment belongs to. Therefore the key number, '801' corresponds to EIC with value, 'M1A1', has the 'Entity-Name', 'Main Battle Tank', is of stereotype, 'Equipment', and belongs to 'Entity Type', 'Equipment'.
  - ◆ 'Equipment-Code' is not recorded in the FDMS database.

➤ Aircraft Table:

- ◆ AIC (Aircraft Identification Code) in the Aircraft table (UOB) can be identified uniquely when combined with the values in 'UIC' (Aircraft table, UOB, Appendix A). AIC is recorded in the Entity table (FDMS) in the 'Entity SDS ID' column.
- ◆ 'Aircraft Description' column in the Aircraft table gives a brief description of the aircraft. The values in the 'Aircraft Description' column are recorded in the 'Entity Name' column in the Entity table.
- ◆ The column 'Entity Stereotype' contains the value 'Equipment' and the column 'Entity Type' contains the value 'Aircraft' identifying that the information inserted in the row is for the Aircraft table. 'Entity Stereotype' and 'Entity Type' are local to the Entity table in the FDMS database.
- ◆ The surrogate key uniquely identifies the entity name of the aircraft, the stereotype and the Unit the aircraft belongs to. Therefore if we have the key number, '1201' it corresponds to AIC with value, 'UH1V', has the 'Entity Name' 'UH1V Utility helicopter', is of stereotype, 'Equipment', and belongs to 'Entity Type', 'Aircraft'.
- ◆ The values in 'Aircraft Code' are not recorded in the FDMS database.

UNIT ORDER OF BATTLE	UNIT ORDER OF BATTLE	Translation Notes	FDMS	FDMS	FDMS	FDMS	FDMS
Table	Attribute		Entity-RDS-ID	Entity-SDS-ID	Entity-Name	Entity-Stereo-type	Entity-Type
Unit	UNIT-IDENTIFICATION-CODE	Per Row		X			
Unit	UNIT-NAME	Per Row			X		
Unit	HOME-NAME	N/A					
Unit	SHIP-CATEGORY	N/A					
		Per Row	SK			Organization	Unit
Personnel	PIC	Per Row		X			
Personnel	PERSONNEL-DESCRIPTION	Per Row			X		
		Per Row	SK			Person	Personnel
Equipment	EIC	Per Row		X			
Equipment	EQUIPMENT-CODE	N/A					
Equipment	EQUIPMENT-DESCRIPTION	Per Row			X		
		Per Row	SK			Equipment	Equipment
Aircraft	AIC	Per Row		X			
Aircraft	AIRCRAFT-CODE	N/A					
Aircraft	AIRCRAFT-DESCRIPTION	Per Row			X		
		Per Row	SK			Equipment	Aircraft

\* SK represents a surrogate Key that uniquely identifies the data entries in the FDMS database tables.

**Table 2.1: Mapping UOB Tables to FDMS Entity Table**

## 2.2.2 Mapping UOB Tables to the Entity-Component Tables in the FDMS

Table 2.2 illustrates which columns in the UOB tables map in the Entity-Component table. The Entity-Component table provides linking information concerning components of entities. Thus it maps to parent-child information from all UOB tables.

### ➤ Unit Table:

- ◆ The ‘Parent-Unit-Identification-Code’ identifies the parent for a unit identified by UIC. In the FDMS Entity tables, units are identified by both RDS-ID and SDS-ID. Thus both components and composites have the same pairs of RDS-ID and SDS-ID that appear in the Entity table. The linkages shown by each row in the Entity-Component table from Component RDS-ID/ SDS-Id pair to Composite RDS-ID/ SDS-ID pair corresponds to the linkage in the Unit table from UIC to Parent Unit Identification Code.
- ◆ Example: A ‘Component-Entity-RDS-ID’ of ‘102’ has a corresponding row with value ‘BN HQ’ of the ‘UIC’ (UOB) in the ‘Component-Entity-SDS-ID’ column in the component table (FDMS), has a Parent-Unit-Identification-Code, ‘ARMR BN’ in the ‘‘Composite Entity SDS ID’’ and has a cardinality of ‘1’. This indicates ‘BN HQ’ is a component of ‘ARMR BN’.
- ◆ All values in the ‘Entity-Component-Cardinality’ column are ‘1’ since the UOB does not provide cardinality information.

### ➤ Personnel Table:

- ◆ The UIC identifies the parent to a personnel row identified by PIC. In FDMS Entity-Component table, personnel rows are uniquely identified by the combination of RDS-ID and SDS-ID. Thus both Components and Composites

have the same pairs of RDS-ID and SDS-ID that appear in the Entity table. The linkages shown by each row in the Entity Component table correspond to the linkage in the Personnel table from PIC to UIC.

- ◆ Example: The 'Component-Entity-RDS-ID' number '402' corresponds to PIC with value, ' MAJ', contains 'BN HQ value in the 'UIC' and has a cardinality of '1'. This indicates that there is one 'MAJ' for 'BN HQ'.

➤ Equipment Table:

- ◆ The UIC identifies the parent to an equipment row identified by EIC. In FDMS Entity-Component Table, equipment rows are uniquely identified by the combination of RDS-ID and SDS-ID. Thus both Components and Composites have the same pairs of RDS-ID and SDS-ID that appear in the Entity table. The linkages shown by each row in the Entity Component table correspond to the linkage in the Equipment table from EIC to UIC.
- ◆ Example: The 'Component-Entity-RDS-ID' number '802' correspond to EIC with value, ' M23', contains UIC, 'BN HQ', and has a cardinality of '1'. This indicates that there is one ('M23') sidearm for BN HQ.

➤ Aircraft Table:

- ◆ The UIC identifies the parent to an aircraft row identified by AIC. In FDMS Entity-Component table, aircraft rows are uniquely identified by the combination of RDS-ID and SDS-ID. Thus both Components and Composites have the same pairs of RDS-ID and SDS-ID that appear in the Entity table. The linkages shown by each row in the Entity Component table correspond to the linkage in the Aircraft table from AIC to UIC.

- ◆ Example: The ‘Component-Entity-RDS-ID’ number ‘1202’ corresponds to AIC with value, ‘UH1V’, contains ‘UIC’, CO HQ and has a cardinality of ‘1’. This indicates that there is one (‘UH1V’) helicopter for CO HQ.

UNIT ORDER OF BATTLE	UNIT ORDER OF BATTLE	Translation Notes	FDMS	FDMS	FDMS	FDMS	FDMS
Table	Attribute		Component-Entity-RDS-ID	Component-Entity-SDS-ID	Composite-Entity-RDS-ID	Composite-Entity-SDS-ID	Entity-Component-Cardinality
Unit	UNIT-IDENTIFICATION CODE	Per Row		X			
Unit	PARENT-UNIT IDENTIFICATION CODE	Per Row				X	
		Per Row	SK		SK		1
Personnel	PIC	Per Row		X			
Personnel	UNIT IDENTIFICATION CODE	Per Row				X	
		Per Row	SK		SK		1
Equipment	EIC	Per Row		X			
Equipment	UNIT IDENTIFICATION CODE	Per Row				X	
		Per Row	SK		SK		1
Aircraft	AIC	Per Row		X			
Aircraft	UNIT IDENTIFICATION CODE	Per Row				X	
		Per Row	SK		SK		1

**Table 2.2: Mapping UOB Tables to FDMS Entity-Component Table**

### 2.2.3 Mapping UOB Attributes to Characteristic Table in the FDMS Database

The FDMS Characteristic tables define extensions to the basic information contained in the Entity table. Once the additional information categories have been defined in the Characteristic table, the actual data can be entered in the FDMS Entity-Characteristic table. Because of this, the Characteristic table has the names of columns from the UOB tables rather than values. Each of the names entered in this table has a surrogate key placed in the 'Character-RDS-ID' that uniquely identifies the rows so that they can be related to the rows of the Entity-Characteristic table.

➤ Unit Table:

- ◆ Column name 'Country-Code' is recorded in the Characteristic-Name, so that the 'Country-Code' values can be entered in the related rows of the Entity-Component table.

➤ Personnel Table:

- ◆ Column names 'Personnel-Quantity-Required' and 'Personnel -Quantity-Authorized' are recorded under the Characteristic-Name column, so that the 'Personnel-Quantity-Required' and 'Personnel -Quantity-Authorized' values can be entered in the related rows of the Entity-Component table.

➤ Equipment Table:

- ◆ Column names 'Equipment-Quantity-Required' and 'Equipment-Quantity-Authorized' are recorded under the Characteristic-Name column so that the 'Equipment-Quantity-Required' and 'Equipment-Quantity-Authorized' values can be entered in the related rows of the Entity-Component table.

➤ Aircraft Table:

- ◆ Column names ‘Aircraft-Quantity-Required’ and ‘Aircraft-Quantity-Authorized’ are recorded under the Characteristic-Name column, so that the ‘Aircraft-Quantity-Required’ and ‘Aircraft-Quantity-Authorized’ values can be entered in the related rows of the Entity-Component table.

UNIT ORDER OF BATTLE	UNIT ORDER OF BATTLE	Translation Notes	FDMS	FDMS	FDMS
Table	Attribute		Characteristic-RDS- ID	Characteristic-SDS -D	Characteristic-Name
Unit		Per Database	SK	Null	COUNTRY-CODE
Personnel		Per Database	SK	Null	PERSONNEL-QUANTITY-REQUIRED
		Per Database	SK	Null	PERSONNEL-QUANTITY-AUTHORIZED
Equipment		Per Database	SK	Null	EQUIPMENT-QUANTITY-REQUIRED
		Per Database	SK	Null	EQUIPMENT-QUANTITY-AUTHORIZED
Aircraft		Per Database	SK	Null	AIRCRAFT-QUANTITY-REQUIRED
		Per Database	SK	Null	AIRCRAFT-QUANTITY-AUTHORIZED

**Table 2.3: Mapping UOB Tables to FDMS Characteristic Table**

#### 2.2.4 Mapping UOB Tables to Entity-Characteristic Table in the FDMS

Table 2.4 illustrates the columns in the UOB tables that map to the columns in the FDMS Entity-Characteristic table. The important point to note in the Entity-Characteristic table is that there are two sets of similar values in the Entity-RDS-ID’ column in the Personnel, Equipment, and Aircraft column. Therefore it is not the data value in this column that

uniquely identifies that row but a combination of values both in 'Entity RDS ID' and 'Characteristic-RDS-ID' (FDMS Entity-Characteristic table, Appendix B). These in turn relate back to the Entity table and Characteristic table, respectively.

➤ Unit Table:

- ◆ The values in 'UIC' column (UOB) are recorded in 'Entity SDS ID' column (FDMS). Once again, as also shown in Tables 2.1 and 2.2, we have a pair of values in the RDS-ID and SDS-ID columns, which were originally defined in the Entity table.
- ◆ The value from the 'Country-Code' Unit table is mapped to the 'Entity-Characteristic-String-Value'.
- ◆ Example: If we have the 'Entity RDS ID' value of '103', the 'Entity SDS ID' is 'ARMR CO', the 'Characteristic-RDS-ID' value is '201' (which relates to the characteristic value "Country-Code" in the FDMS Characteristic table), with a 'Entity-Characteristic-String-Value' as 'US', which means the Country-Code is US for 'ARMR CO'.

➤ Personnel Table:

- ◆ The values in 'PIC' column (UOB) are recorded in 'Entity SDS ID' column (FDMS). Once again, as also shown in tables 2.1 and 2.2, we have a pair of values in the RDS-ID and SDS-ID columns, which were originally defined in the Entity table.
- ◆ The values in 'Entity RDS ID' and the 'Characteristic-RDS-ID' columns together identify the row in the Entity-Characteristic table.

- ◆ Example: ‘ If we have Entity-RDS-ID’ value of ‘403’, the ‘Entity SDS ID’ is ‘DRIVER’, the Characteristic-RDS-ID value is ‘601’, which relates to the ‘Characteristic’ column ‘Personnel-Quantity-Required’ (FDMS Characteristic table) and, ‘Entity-Characteristic-Numeric-Value’ of ‘1’, meaning that one driver is required for this unit.

Also, if we look at the second set of entries in the same table with ‘Entity RDS ID’ value of ‘403’, most of the values remain the same, except the ‘Characteristic-RDS-ID’ value is now ‘701’, which refers to Personnel-Quantity-Authorized, with ‘Entity-Characteristic-Numeric-Value’ of ‘1’, meaning that one driver is authorized for this unit.

➤ Equipment Table:

- ◆ The values in ‘EIC’ column (UOB) are recorded in ‘Entity SDS ID’ column (FDMS). Once again, as also shown in tables 2.1 and 2.2, we have a pair of values in the RDS-ID and SDS-ID columns, which were originally defined in the Entity table.
- ◆ The values in ‘Entity RDS ID’ and the ‘Characteristic-RDS-ID’ columns together identify the row in the Entity-Characteristic table.
- ◆ Example: ‘ If we have Entity-RDS-ID’ value of ‘803’, the ‘Entity SDS ID’ is ‘M1R’, the Characteristic-RDS-ID value is ‘1001’, which relates to the ‘Characteristic’ column ‘Equipment-Quantity-Required’ (FDMS Characteristic table) and, ‘Entity-Characteristic-Numeric-Value’ of ‘6’, meaning that six M1 rifles are required for this unit.

Also, if we look at the second set of entries in the same table with ‘Entity RDS ID’ value of ‘803’, most of the values remain the same except the ‘Characteristic-RDS-ID’ value is now ‘1101’, which refers to Equipment-Quantity-Authorized, with ‘Entity-Characteristic-Numeric-Value’ of ‘6’, meaning that six M1 rifles are authorized for this unit.

➤ Aircraft Table:

- ◆ The values in AIC column (UOB) are recorded in ‘Entity SDS ID’ column (FDMS). Once again, as also shown in tables 2.1 and 2.2, we have a pair of values in the RDS-ID and SDS-ID columns, which were originally defined in the Entity table.
- ◆ The values in ‘Entity RDS ID’ and the ‘Characteristic-RDS-ID’ columns together identify the row in the Entity-Characteristic table.
- ◆ Example: ‘ If we have Entity-RDS-ID’ value of ‘1201’, the ‘Entity SDS ID’ is ‘UH1V’, the Characteristic-RDS-ID value is ‘1401’, which relates to the ‘Characteristic’ column ‘Aircraft-Quantity-Required’ (FDMS Characteristic table) and, ‘Entity-Characteristic-Numeric-Value’ of ‘1’, meaning that there is one UH1V helicopter required for this unit.

Also, if we look at the second set of entries in the same table with ‘Entity-RDS-ID’ value of ‘1201’, most of the values remain the same except the ‘Characteristic-RDS-ID’ value is now ‘1501’, which refers to Personnel-Quantity-Authorized, with ‘Entity-Characteristic-Numeric-Value’ of ‘1’, meaning that there is one UH1V helicopter authorized for this unit.

UNIT ORDER OF BATTLE	UNIT ORDER OF BATTLE	Translation Notes	FDMS	FDMS	FDMS	FDMS	FDMS	FDMS
Table	Attribute		Entity- RDS-ID	Entity- SDS- ID	Character- istic RDS ID	Character- istic- SDS-ID	Entity- Character -istic Numeric Value	Entity Character- istic String Value
Unit	UNIT- IDENTIFICA- TION-CODE	Per Row		X				
Unit	COUNTRY CODE	Per Row						X
		Per Row	SK		SK	Null		
Personnel	PIC	Per Row		X				
Personnel	PERSONNEL QUANTITY REQUIRED	Per Row					X	
Personnel	PERSONNEL QUANTITY AUTHORIZED	Per Row					X	
		Per Row	SK		SK	null		
Equip- ment	EIC	Per Row		X				
Equip- ment	EQUIPMENT QUANTITY REQUIRED	Per Row					X	
Equip- ment	EQUIPMENT QUANTITY AUTHORIZED	Per Row					X	
		Per Row	SK		SK	null		
Aircraft	AIC	Per Row		X				
Aircraft	AIRCRAFT QUANTITY REQUIRED	Per Row					X	
Aircraft	AIRCRAFT QUANTITY AUTHORIZED	Per Row					X	
		Per Row	SK		SK	null		

**Table 2.4: Mapping UOB Table to FDMS Entity-Characteristic Table**

Table 2.5 shows the entries in the FDMS tables for each entry in the UOB tables. It does not show the FDMS surrogate keys assigned to the RDS-ID columns.

<b>UOB</b>	<b>Table Entity (FDMS)</b>	<b>Table Entity Component (FDMS)</b>	<b>Table Entity- Characteristic (FDMS)</b>
<b>Table Unit</b>			
Unit-Identification-Code (Table Unit)	Entity-SDS-ID	Component-Entity- SDS-ID	Entity-SDS-ID
Parent-Unit- Identification-Code		Composite-Entity-SDS- ID	
Unit-Name	Entity-Name		
Country-Code			Entity-Characteristic-String- Value
<b>Table Personnel</b>	<b>Table Entity (FDMS)</b>	<b>Table Entity Component (FDMS)</b>	<b>Table Entity- Characteristic (FDMS)</b>
PIC	Entity-SDS-ID	Component-Entity- SDS-ID	Entity-SDS-ID***
Unit-Identification-Code		Composite-Entity-SDS- ID	
Personnel-Description	Entity-Name		
Personnel Quantity Required			Entity-Characteristic- Numeric-Value***
Personnel Quantity Authorized			Entity-Characteristic- Numeric-Value***
<b>Table Equipment</b>	<b>Table Entity (FDMS)</b>	<b>Table Entity Component (FDMS)</b>	<b>Table Entity- Characteristic (FDMS)</b>
EIC	Entity-SDS-ID	Component-Entity- SDS-ID	Entity-SDS-ID***
Unit-Identification-Code		Composite-Entity-SDS- ID	
Equipment-Description	Entity-Name		
Equipment Quantity Required			Entity-Characteristic- Numeric-Value***
Equipment Quantity Authorized			Entity-Characteristic- Numeric-Value***
<b>Table Aircraft</b>	<b>Table Entity (FDMS)</b>	<b>Table Entity Component (FDMS)</b>	<b>Table Entity- Characteristic (FDMS)</b>
AIC	Entity-SDS-ID	Component-Entity- SDS-ID	Entity-SDS-ID***
Unit-Identification-Code (Table Aircraft)		Composite-Entity-SDS- ID	
Aircraft-Description	Entity-Name		
Aircraft Quantity Required			Entity-Characteristic- Numeric-Value***
Aircraft Quantity Authorized			Entity-Characteristic- Numeric-Value***

**Table 2.5: Mapping UOB Records to Multiple Entries in FDMS**

### 2.2.5 Tracing and Updating the Database

In order to update the database, the entries are first made in the Entity-Component table, then in the Entity table and finally in the Entity-Characteristic table. No entries will be necessary for the Characteristic table.

#### **Component Table:**

- ◆ Start with the Entity-Component table.
- ◆ Check to see whether the incoming ‘AIC’ and ‘Unit-Identity-Code’ already exist.
- ◆ If it does not, enter them in the ‘Component Entity SDS ID’ and the Composite-Entity- SDS-ID respectively.
- ◆ The ‘Composite-Entity-RDS-ID is obtained from the Entity-Component table, based on the value of the ‘Composite Entity SDS ID’.
- ◆ The Cardinality is maintained as ‘1’.

#### **Entity Table:**

- ◆ A new record is then inserted into the Entity table.
- ◆ The ‘Entity RDS ID’ and the ‘Entity SDS ID’ values correspond to the ‘Component Entity-RDS-ID’ and the “‘Component Entity SDS ID’” values in the Entity-Component table.
- ◆ The ‘Entity Name’ is derived from the ‘Aircraft/Equipment/Personnel-Description’ in the incoming message object.
- ◆ The ‘Entity-Stereotype’ and the ‘Entity Type’ are deduced from the identity of the UOB table created (Equipment/Personnel /Aircraft).

**Entity-Characteristic Table:**

- ◆ For each entry, the ‘Entity RDS ID’ and the ‘Entity SDS ID’ correspond to the values in the Entity table.
- ◆ The ‘Characteristic-RDS-ID’ and the ‘Characteristic-SDS-ID’ are obtained from the FDMS Characteristic table, corresponding to the UOB table created for the ‘Equipment/Personnel /Aircraft Quantity Required’ and ‘Equipment/Personnel /Aircraft Quantity’ Authorized.
- ◆ The numeric data for each of these is contained within the incoming data in the ChangeObj object.

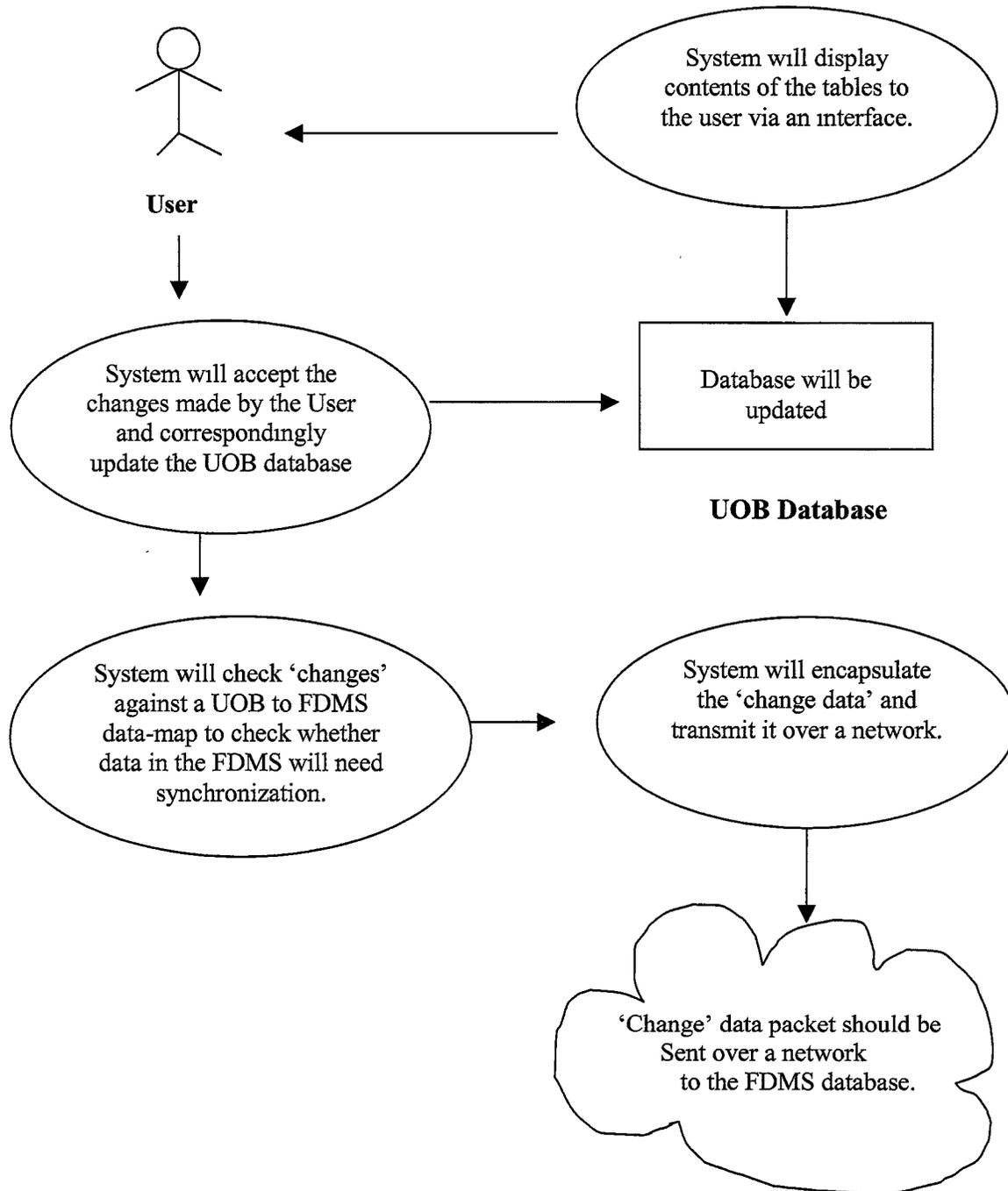
**2.3 Systems Specification**

Recalling our objective, it is to synchronize data in diverse databases, located geographically apart. Hence, our system will have two components, one on the transmission side (UOB) and the other on the receiving side (FDMS).

**2.3.1 Transmission side (UOB)**

- ◆ A data entry application with a user interface will display the values in the various tables of the UOB database and interactively allow a user to make changes to the database. Figure 2.1 illustrates this application.
- ◆ The system will accept the changes made by the user.
- ◆ The system will correspondingly update the UOB database.
- ◆ The system will check the modifications made against a UOB to FDMS data map to determine whether modified data is present in the FDMS database and will need to be synchronized.

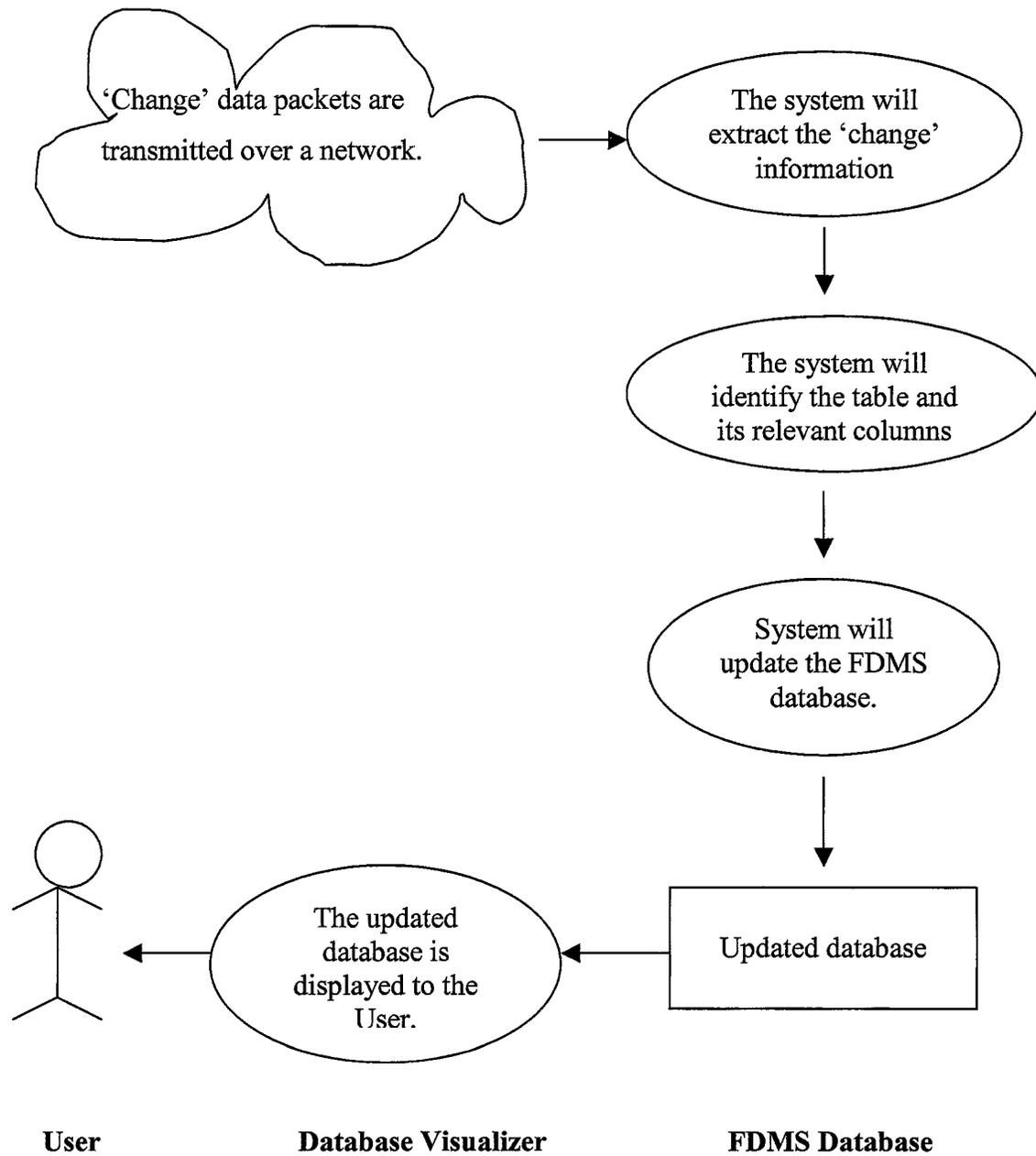
- ◆ The system will encapsulate the 'change' information (identify table, database columns and values) in a format that will enable its transmission over the network to the receiving system.
- ◆ The system will communicate with a standard communication protocol on the network to transmit the packets encapsulating the 'change data'.



**Figure 2.1: Requirements Specification -UOB Side**

### 2.3.2 Receiving Side (FDMS)

- ◆ The system will communicate with the network to receive the 'change data' information. Figure 2.2 illustrates the receiving side activities
- ◆ The system will extract the 'change' information from the communicated packets.
- ◆ The system will decode the information to identify the FDMS table and relevant columns that need updating with the data received.
- ◆ The system will update the FDMS database.
- ◆ The system will display the updated tables to the FDMS user.



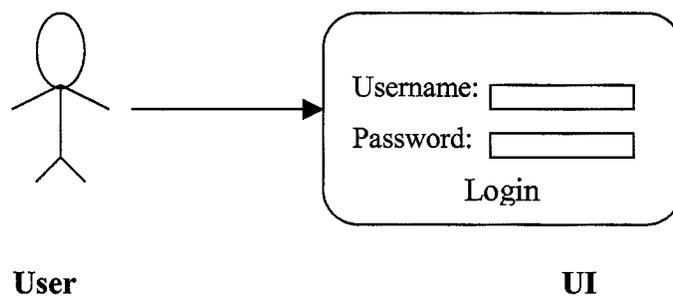
**Figure 2.2: Requirements Specification – FDMS Side**

## 2.4 Stages of Operation

Following are the sequence of stages and steps that the system will support for its operation.

### ◆ User Login

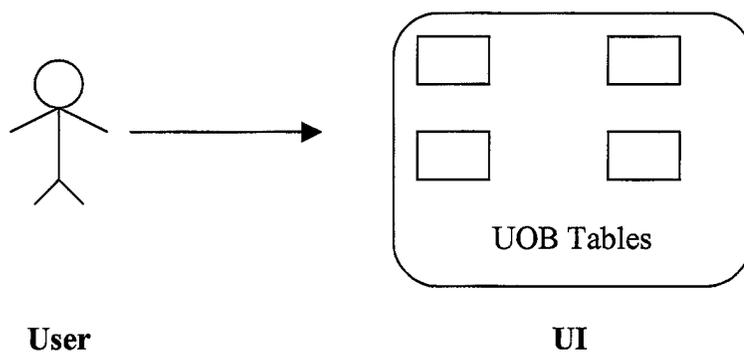
The user will initiate the process by requesting a change to the UOB database through the interactive user interface (UI). The user will be able to access the UI by entering a valid username and password. Figure 2.3 illustrates the user login.



**Figure 2.3: Login Page**

### ◆ Accessing the Database Tables

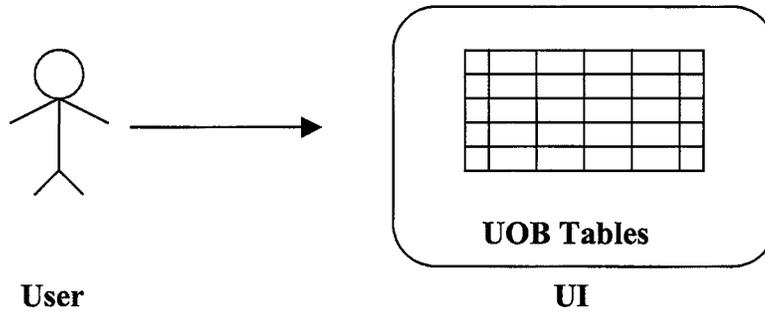
The user will request access of UOB tables in the UOB database to edit the required data. Figure 2.4 illustrates accessing the UOB tables.



**Figure 2.4: Database Access**

◆ **Editing the tables**

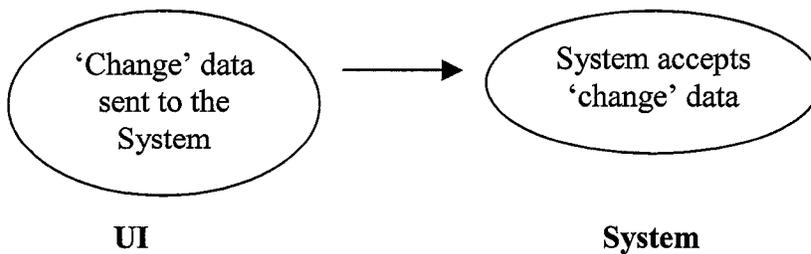
The user will edit the data in the required UOB table. Figure 2.5 illustrates the user editing the database.



**Figure 2.5: Editing a UOB Table**

◆ **Edited tables are forwarded to the system**

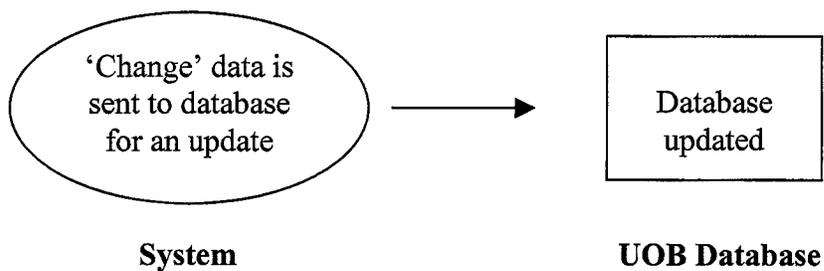
The edited data will be sent to system for further processing. Figure 2.6 illustrates the data being sent from the User Interface to the system for further processing.



**Figure 2.6: Processing the Data**

◆ **‘Change’ data sent to the UOB database**

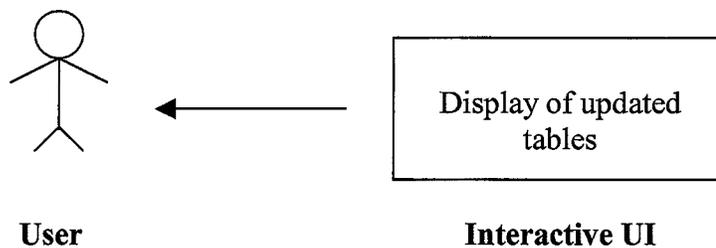
The ‘change’ data request will be communicated to the database by our system and the corresponding changes will be made. Figure 2.7 illustrates the UOB database being updated.



**Figure 2.7: Updating UOB Database**

◆ **Change confirmation**

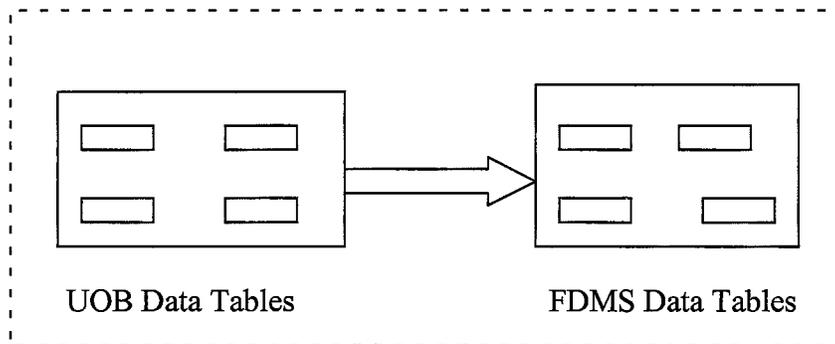
The changes will be relayed back to the UI by our system, and will be made visible to the user. Figure 2.8 illustrates the updated UOB database.



**Figure 2.8: Updated Database Displayed**

◆ **Edited UOB data mapped to FDMS data**

The 'change' data that is changed in the UOB database is checked in the common data table to find if there is a requirement to update the database (figure 2.9).

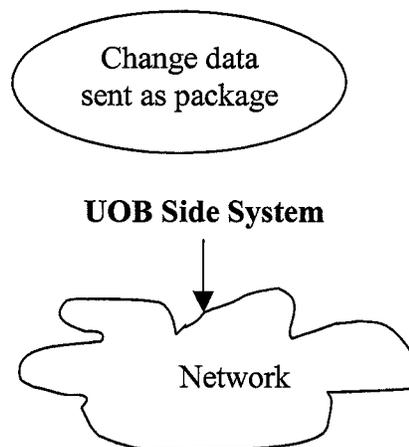


**Common Data Mapping**

**Figure 2.9: Data Mapping**

◆ **'Change' data packaged to FDMS-side**

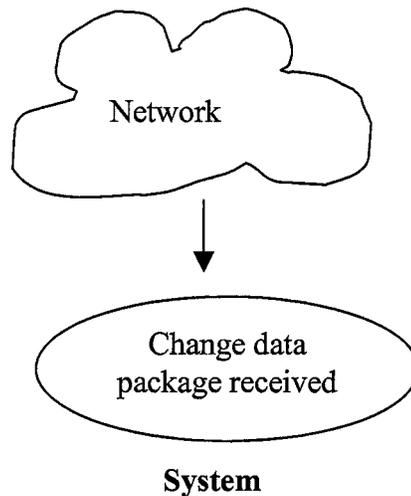
The modifications, if made to data common with the FDMS database, will be encapsulated and transmitted over the network to the FDMS system. Figure 2.10 illustrates the change data being transferred via a network to the FDMS side.



**Figure 2.10: Data Transferred from UOB Database**

- ◆ **'Change' data accepted by the system on the FDMS side**

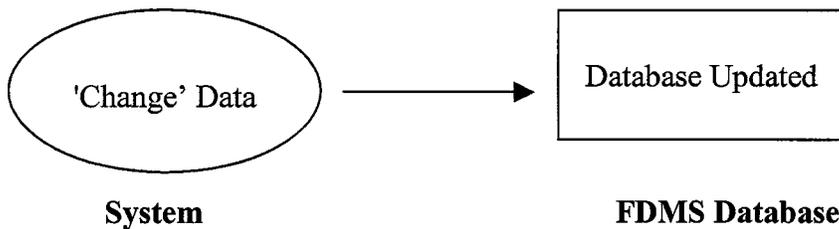
The FDMS-side system will receive the packets, decode them and extract the 'change data' information. Figure 2.11 illustrates the change data being received via a network from the FDMS side.



**Figure 2.11 Data Transfer to FDMS Database**

- ◆ **'Change' data is used by the system to update the FDMS Database.**

'Change' data is sent to the FDMS database for an update. Figure 2.12 illustrates the change data being transferred to the FDMS database.

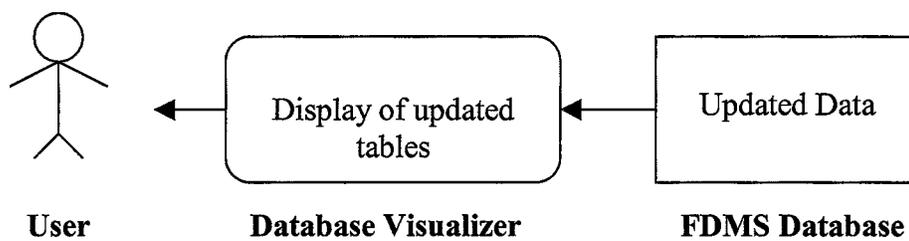


**Figure 2.12 FDMS Database Updated**

◆ **Updated FDMS database displayed**

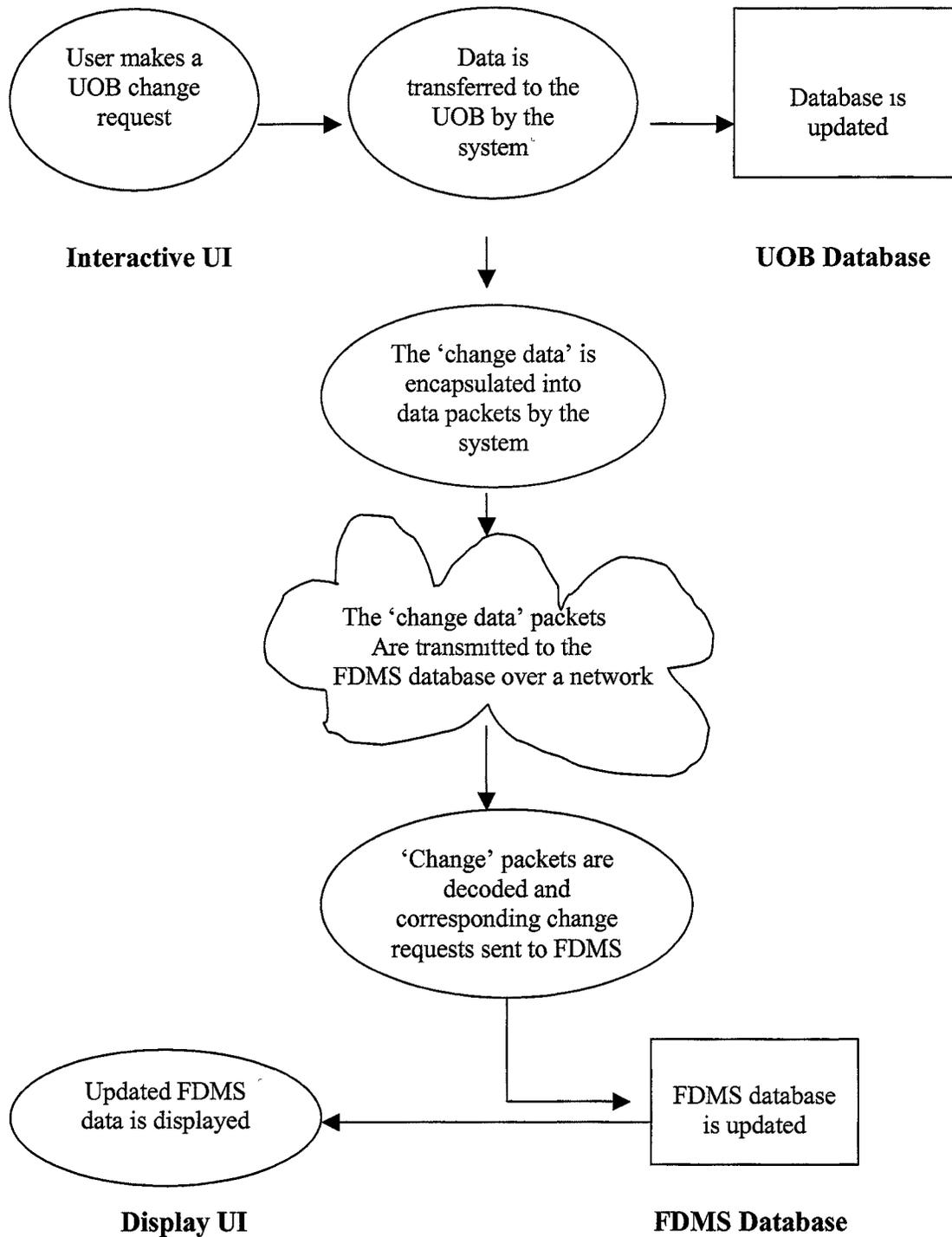
The system receives UOB changes whenever they are made and automatically updates the FDMS, displaying the changes the next time the FDMS user logs in.

Figure 2.13 illustrates the updated data being displayed to the user through the database visualizer.



**Figure 2.13 FDMS Tables Displayed**

- ◆ Figure 2.14 summarizes the stages of operation as per our systems requirements.



**Figure 2.14 Stages of Operation**

## **2.5 Summary of the Chapter**

In this chapter the systems requirements are defined in detail including a step-by-step description of the stages of operation.

In the next chapter we will discuss the three distributed enterprise technologies that were available to us to meet our functional requirements and discuss the reasons for their adoption/rejection. We will also describe the technology (J2EE) we have selected in detail, and show how our system is built using this technology and an n-tier system architecture.

## CHAPTER 3

### ENABLING TECHNOLOGIES AND OUR SYSTEM

#### 3.1 Choosing the Enabling Technologies

Among the several technologies employed in this project, including programming language, database management system, operating system, the most significant choice is the selection of the distributed enterprise technology. We looked in detail at several contemporary technologies in making this important choice.

#### 3.2 Choice of Distributed Enterprise Technology

To build our system, we would prefer to use technology that is:

- ◆ An Industry Standard – We would like to avoid proprietary technology and use something widely accepted and defined as a standard by the industry.
- ◆ Widely in use – We would like to use technology that has been extensively adopted and is likely to be in use for several more years. Wide adoption will give us a good choice of reference material and several human references we might be able to call upon.
- ◆ Relatively inexpensive – Any system is only as good as its practical viability. We would like to use technology that is relatively affordable and easily licensable.

Based on our requirements, we analyzed three technologies: Sequoia's XML Portal Server, Microsoft's .Net, and Sun's Java 2 Enterprise Edition. We discuss each of these below in our quest for the best distributed enterprise technology solution.

### **3.2.1 XML Portal Server (XPS)**

XML Portal Server (XPS) was developed by, Sequoia Software Corporation. An industrial collaborator was to help with requirements for applications and agreed to make XPS available.

XPS is a portal platform that features an extensible, standards-based architecture and enables true interactivity with diverse and disparate systems. It is portrayed as a new media communication vehicle that extends information and processes and communicates resources to the many constituencies that make up an organization's business environment. As such, it seems to have all the capabilities needed by our system for resource management, communication to the network and transmitting targeted data over the network [XPS02].

XPS provides a comprehensive set of functionalities needed to customize, and manage a portal and provides a bi-directional interface to the systems that feed it, giving it the ability to access not only relevant information, but also update, edit, and act on information of the portal. It is inherently based on XML and has the ability to understand the context of the information flowing through the portal. Through its use of XML, XPS has the ability to understand the context of the information flowing through the portal. Within the portal server, XML becomes a powerful mechanism for building contextual indexes that streamline information discovery. XML provides the basic architecture for scalability, high throughput and flexibility. When applied to the interchange of

information between these systems, XML plays a pivotal role in supporting these expanding capabilities. XML is mainly important to facilitate information interchange. XML supports machine-to-machine processing, are readily transformed into Web pages for consumption by end users, and are easily transportable between disparate applications.

The software was found to be inappropriate because:

- ◆ It is a proprietary technology.
- ◆ There are very few references available for its installation and use.
- ◆ The software is expensive.
- ◆ Following a merger, active sales efforts were discontinued. Obtaining technical support became a problem.
- ◆ Alternative software is freely available and less complicated and
- ◆ Obtaining support for using the software is difficult and complicated.

### 3.2.2 .Net

. Net is the latest architecture from Microsoft for building distributed enterprise applications. It provides enhanced interoperability features based upon open Internet standards. Microsoft's .Net is a platform built on top of the operating system. It is language independent, but platform dependent. Programmers can develop in the Windows platform. It can be used in 27 languages, including C#. The applications are compiled to Microsoft Intermediate Language code, which is a low-level platform-independent language [TPV02]. .Net provides the following services:

- ◆ **The runtime environment** –.Net consists of a runtime environment called the common language runtime and a set of supporting libraries. The runtime

environment controls the installation, loading, and execution of .NET applications.

The libraries provide code for common programming tasks.

- ◆ **ASP.NET** – ASP.Net is a new environment that runs on Internet Information Services and makes it much easier for programmers to write code that constructs HTML-based Web pages for browser viewing. ASP.Net features a new language-independent way of writing code and tying it to Web page requests.
- ◆ **XML Web services** - .Net framework provides a set of services that allows a server to expose its functions to any client on any machine running any operating system. A set of functions exposed this way is called Web services.
- ◆ **Support for handling XML documents** - .Net framework contains outstanding support for writing applications that handle XML documents and streams.  
[PDS03].
- ◆ **Connection from one .Net system to another** – The .Net remoting API allows programmers to write code that creates objects and calls functions from one .Net system to another .Net System. .Net therefore provides a complete host of services and capabilities suitable for our system. References and manuals are extensively available [PDS03].

However, we decided not to use this technology for the following reasons:

- ◆ It is relatively new and unproven. Given the history of its manufacturer with respect to new software, we found it advisable to wait until its imperfections were found and corrected.
- ◆ Although well on its way to becoming a standard, there are few practical implementations yet.

- ◆ . Net is much cheaper than XPS, but there is still a cost associated with the product.

### 3.2.3 Java 2 Enterprise Edition (J2EE)

Java 2 Enterprise Edition (J2EE) is licensed by Sun Microsystems. Its technology provides a component-based approach to the design, development, assembly and deployment of enterprise applications. The Java 2 computing platforms are Java language-centric but have a platform-independent architecture. It can run on several different operating systems, including Solaris, Unix, Windows, Linux, and MacOS. The J2EE supports a multitiered (n-tier) approach:

- ◆ The client tier, which manages the client interface and communicates with the web and business tiers (J2EE) to obtain the information needed to present to the user.
- ◆ The web tier, which is responsible for creating the presentation used by the client to interact with the user.
- ◆ The business tier reacts to client calls and retrieves data from the database tier.
- ◆ The database tier, which consists of data tables, implemented as a relational database.

J2EE was found to have all the capabilities we need for our system. It has a programmable base in a widely used programming language, has user interface design tools, has a built in interface with the Ethernet for communication and has a messaging system for communication with other systems. Additionally, it has the following benefits:

- ◆ It is widely in use. There are thousands of systems using this technology.
- ◆ It is an industry standard.
- ◆ It has multiple vendors – it is not proprietary.
- ◆ It has plenty of resources and references.

- ◆ Some of its vendors offer open source versions with free licenses.

Based on all of these benefits, we decided to use J2EE as the distributed enterprise technology for our system. The technology and its architecture are described in much greater detail in the following chapters.

### **3.3 Other Supporting Technologies**

Of the technologies employed include an operating system, a programming language, and a database management system.

#### **3.3.1 Operating System and J2EE**

As mentioned earlier since J2EE is platform independent we could select from Solaris, Windows, MacOS, Linux operating system. We decided to use the Windows operating system since it is the most convenient to use.

#### **3.3.2 Programming Language and J2EE**

J2EE is language-centric. Java is the only programming language compatible with j2EE.

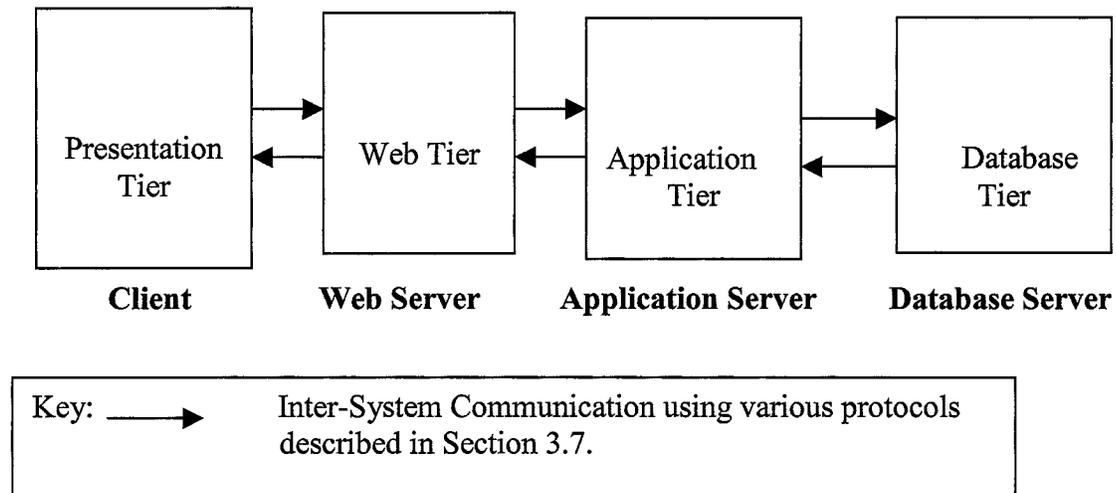
#### **3.3.3 Database Management System and J2EE**

Oracle and MySQL are two database management systems that could have been used. We selected MySQL since it could be freely downloaded, was easy to handle and was compatible with the J2EE server.

### **3.4 Java 2 Enterprise Edition (J2EE) and Our System**

J2EE is an n-tier technology that separates the presentation, business logic, and data storage into separate tiers. Since tiers are logical concepts, it is actually the combination of various servers that creates the system. Figure 3.1 illustrates the tiers and the servers. We therefore have the presentation tier on the client, the web tier on the web server, the

logic (application) tier on the application server and the database tier on the database server [SSJ02].



**Figure 3.1: J2EE Architecture**

The design of our system is based on this separate tier technology. Our system will logically be made of the four tiers and will contain the four software modules illustrated in the above figure.

### **3.4.1 Presentation Tier**

The presentation tier on the client manages the client interface and communicates with the web tier to obtain the information needed to present to the user.

Our system will be using this tier, implemented using a web browser, so that the user can interactively communicate with the components of the system to modify and view the contents of the database.

### **3.4.2 Web Tier**

The web tier on the web server is a program that dishes out web pages in response to requests from a user sitting at a web browser – on the presentation tier. Besides serving static HTML pages they run programs in response to user request and return the dynamic results to the users browsers [JT02].

Our system will use the web tier to send database data to the presentation tier, and to accept data edited by the user to forward it to the application-server, for further processing.

### **3.4.3 Application Tier**

The application tier contains the business logic in the system. All program logic and rules reside here. Typically, user data is provided to this tier by the web tier, is processed here and is stored in the database tier. For data recalled by the presentation tier, this tier obtains the relevant data columns from the database, provides any necessary processing, and returns it via the web tier [JT02].

We will use the application tier on the UOB system to update the UOB database, to verify whether columns changed in the UOB need synchronization in the FDMS database

and if so, to package the 'change' data and communicate it to the FDMS system with appropriate and adequate information, to synchronize the FDMS database.

On the FDMS system, the application server will receive the 'change' data packets from the network, decode them and use the information to update the FDMS database.

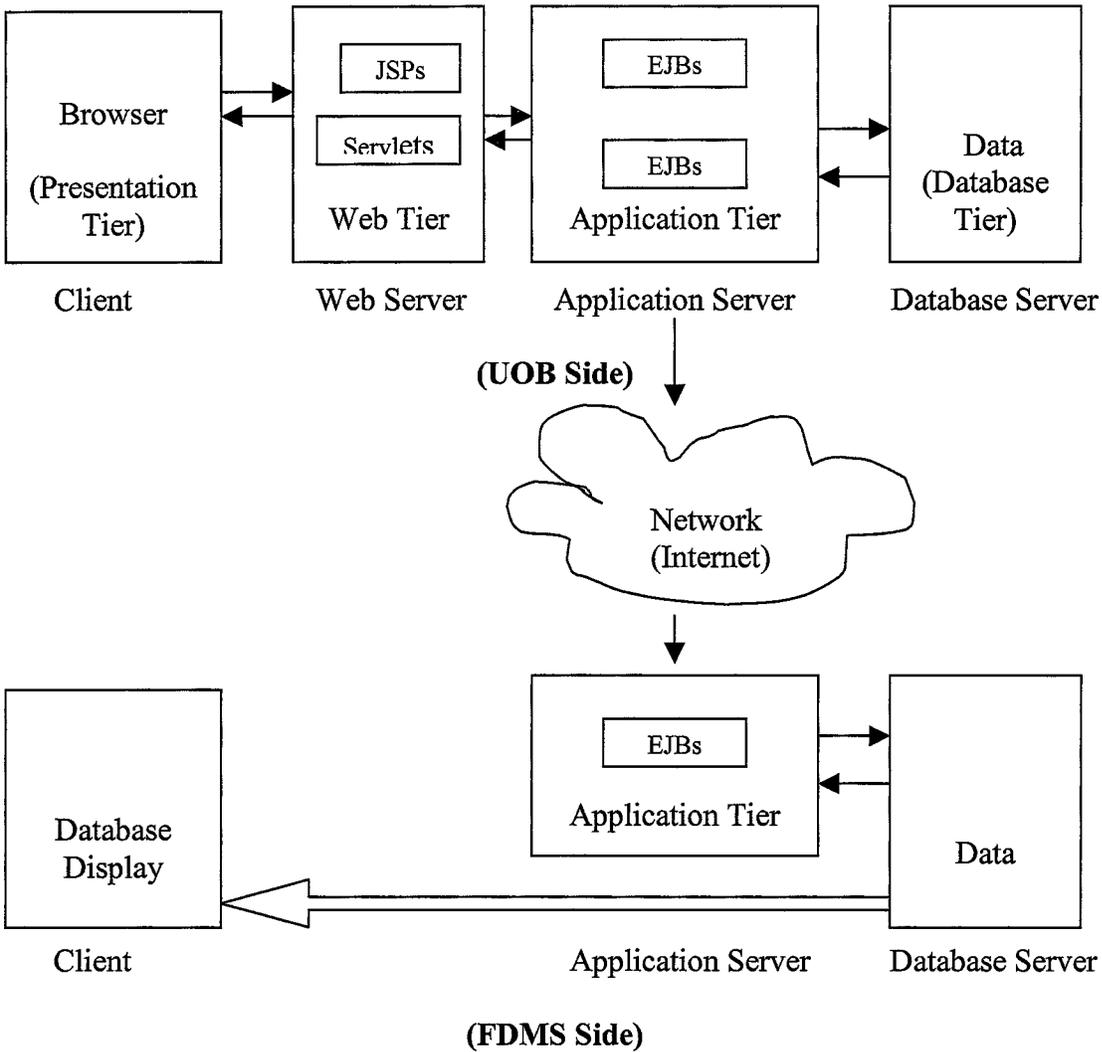
#### **3.4.4 Database Tier**

The database tier on the database server stores all the required data. It is also responsible for maintaining the consistency of application data.

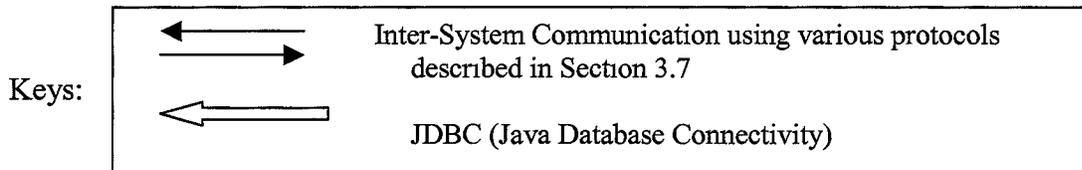
The database tier on our system will store UOB data on the UOB-side and FDMS data on the FDMS-side.

### **3.5 J2EE Components and Our System**

The J2EE system is composed of numerous components, each performing a specific function. They are application-level software units with related classes and files that communicate with other related components in the system. These components can be invoked remotely and can accept and receive parameters and in turn, return specific values. Examples of components are Java Server Pages (JSPs) and Servlets (web components) on the web tier and Enterprise JavaBeans (EJBs) (application components) on the application tier [SSJ02]. Figure 3.2 illustrates the distribution of J2EE components between the UOB processor and the FDMS processor.



\*\*Web Tier is not required on the FDMS side since there is no user interaction



**Figure 3.2: J2EE Components**

In our system, we will be using JSPs and servlets (components) in the Web tier and EJBs in the application-server tier. It is therefore important to understand these components.

### 3.5.1 Web Components and Our System.

- ◆ **Java Server Pages** - JSPs typically generate the user interface for a web-based application. They accept user requests and generate corresponding responses. A typical JSP page appears very similar to an HTML page, with the added ability of displaying dynamic content, typically obtained from servlet components in the system. Internally the Web Server converts each JSP page into a servlet of its own. However, JSPs differ from servlets in their programming model. A JSP page is primarily a formatting document that presents dynamic content, unlike a servlet, which is a program that produces dynamic content. [TPV02].
- ◆ **Servlets** - Servlets are Java programming language classes that dynamically process requests and construct responses. They are programs that run on a web server and are generally invoked by JSPs. The job of the servlet is to read data sent by the user, look up any other information about the request, generate results by talking to the application-tier or database and send the documents back to the client through a JSP. We will be using JSPs in our system to present the pages that the user will view in the presentation tier. Additionally, the JSPs will receive UOB 'change' data from the client application (browser). The JSPs, in turn, will forward this data from the user to appropriate servlets in the system for communication with the application-tier. When the data is processed and the application-tier returns data to the web-tier, it will be received by another servlet, to be forwarded to the appropriate JSP for presentation to the user.

### 3.5.2 Application-Server Components and Our System

- ◆ **Enterprise Java Beans** - Enterprise JavaBeans (EJBs) help developers create business objects that consist of columns and methods that implement business logic. They perform specific tasks by themselves, or forward operations to other enterprise beans. EJBs are under control of a pplication servers[ANAN02]. There are three types of EJBs. However, we will describe only two types that we will require to build our system:

- ◆ **Session beans.** These EJBs perform a task for a client inside the application server. Each bean consists of several methods that contain logic and are invoked for performing specific functions. There are two types of session beans – Stateless and Stateful. Stateless session beans do not retain any state data of their own - it is typically passed in by the calling client. When the client finishes executing this bean, its data is gone. These are highly efficient to use and are the most common types of EJBs used. The application server maintains a pool of these beans and one is allocated out of the pool to a calling client. Stateful session beans contain attributes that retain state information. [AnAn02].

We will be using stateless session beans on the application tier of the UOB-side system to process data.

- ◆ **Message-Driven Beans (MDBs)** - MDBs are components that receive inbound messages delivered via the Java Message Service. These beans contain business logic for handling messages received from communicating clients, typically on other systems[TPV02]. Each bean implements a JMS message listener interfaces, and asynchronously consumes messages sent to a JMS queue that it is listening on. A

message sent to that queue typically wakes up a waiting MDB and invokes the ‘onMessage’ method of that bean. This method contains the business logic required to process the message. A communicating client never accesses a message driven bean directly when a message arrives; it is the application server that calls the message-driven bean’s ‘onMessage’ method to process the message.

We will be using message-driven beans on the FDMS-side of our system to receive and process messages sent by the UOB-side. Each message will contain appropriately formatted ‘change’ data and will contain information about the data that will need synchronization in the FDMS database.

### **3.6 J2EE Containers and Our System**

Containers are standardized runtime environments that provide specific services to the web and Application components. They provide a way for services to be “injected” into the operations of the components without the component developer needing to write specific code. These containers are transparent to the users, and they provide some of the services provided by operating services. They are the interface between a component and the low-level platform-specific functionality that supports the component. Before a web or an enterprise bean component can be executed; it must be assembled into an application and deployed into its container [TPV02].

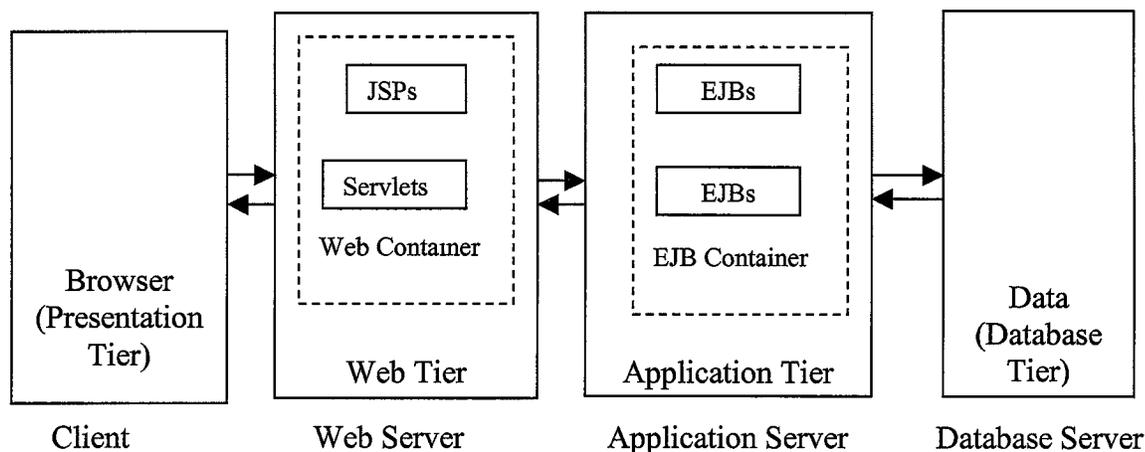
#### **3.6.1 Web Container and Our System**

The web container manages the execution of JSP and servlet components for J2EE applications. The web components and their container run on the web server (figure 3.3). The web container in our system, as defined above, will manage and serve the execution of its JSPs and servlets.

### 3.6.2 EJB Container and Our System

The EJB container manages the execution of Enterprise Java Beans (EJBs) for J2EE applications. Enterprise java beans and their container run on the application server (figure 3.3).

In our system the EJB container will handle the execution of the EJBs.

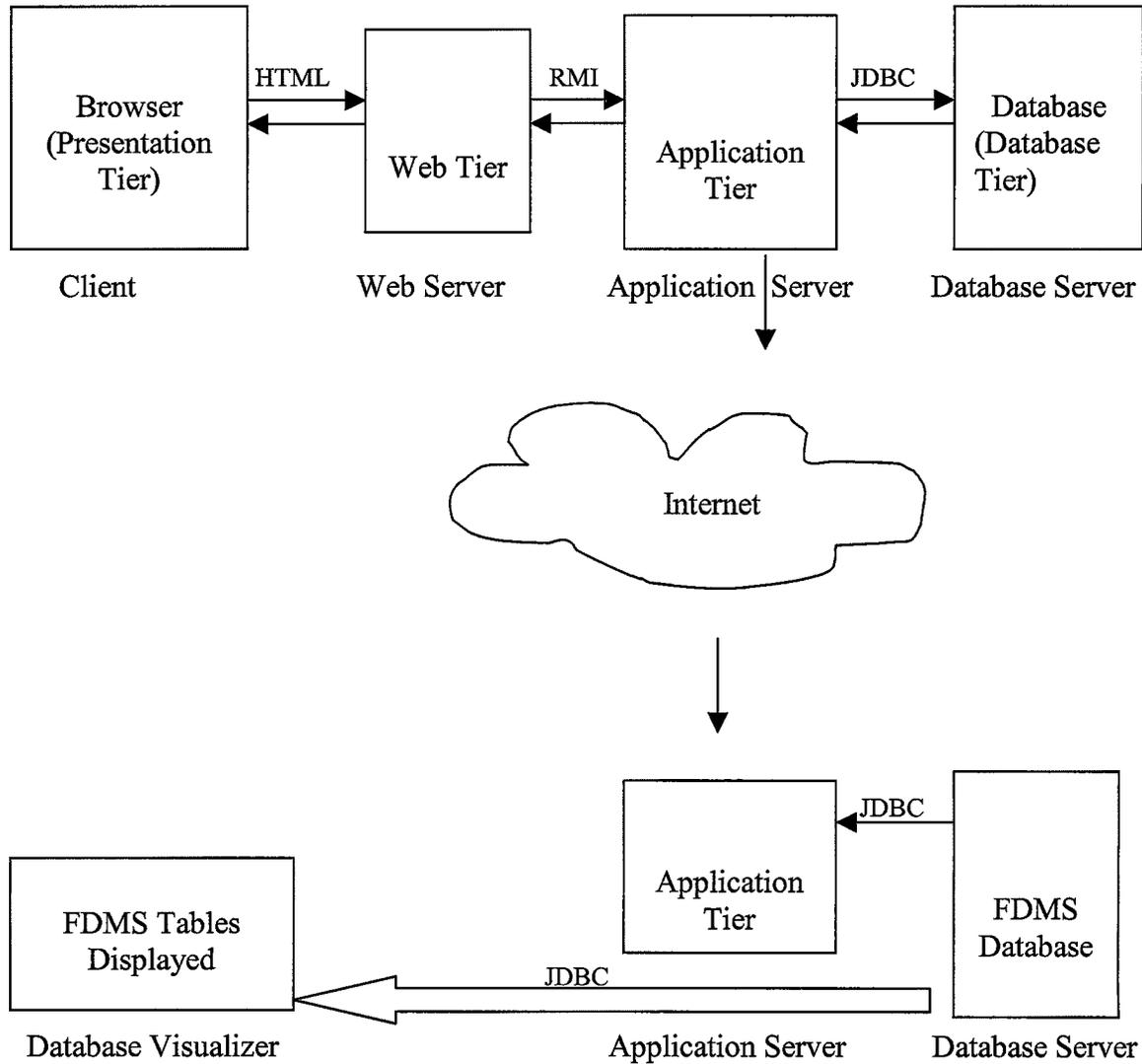


**Figure 3.3: J2EE Containers**

### 3.7 J2EE and Communication

J2EE systems include communication-enabling technology, to enable clients to communicate with J2EE components, and for J2EE components to communicate with each other (figure 3.4), by providing the necessary underlying infrastructure.

The communicating components can be on the same or different computers, the one exception being the Database Visualizer communication with the database server, which must be on the same computer.



**Figure 3.4: J2EE Standard Protocols**

A few of the Java technologies in the communication enabling technology group that we have used to build our system are described below. These are categorized into Communication services, Enterprise services and Internet services. All these services help communicate in different areas of the technology, depending on the area where the service is required.

### 3.7.1 Communication Services

These include the RMI IIOP services and the Java Message Service.

- ◆ **Remote Method Invocation and Inter-ORB Protocol (RMI-IIOP)**

RMI-IIOP is a transparent layer and has the ability to invoke methods on objects located on different computers as easily as invoking a method on objects on the same computer. It is a protocol for invoking services and programs on separate servers [1].

We use the RMI IOP feature on our system to communicate between tiers.

- ◆ **Java Message Service**

Java Messaging Service (JMS) provides a standard unified message API for different message formats to encapsulate and exchange asynchronous messages. The application uses the underlying Enterprise Messaging System to create messages and to communicate asynchronously with one or more peer applications.

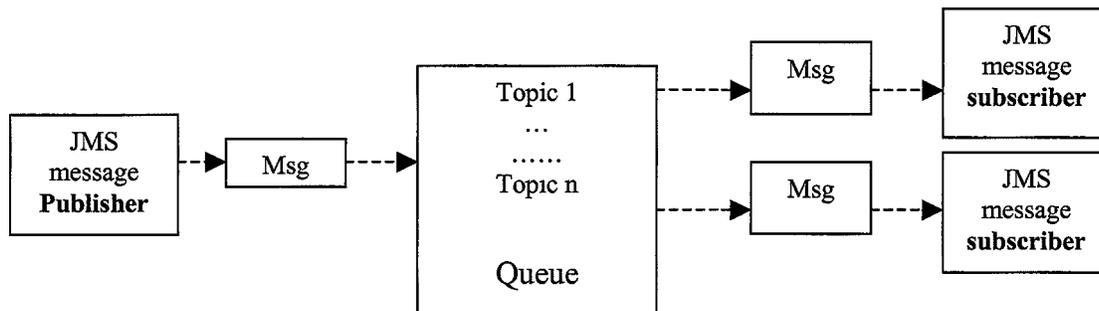
JMS supports the Publish / Subscribe messaging model (defined below). This messaging model depends on the JMS provider, which is a server that provides messaging services to both the producer and the receiver of a given message.

- **Publish / Subscribe**

In publish/subscribe messaging, the publisher publishes messages to a topic destination and subscribers consume the messages from the topic destination (figure 3.5). Publish/subscribe messaging systems provide one-to-many delivery of messages. For example, when a message is published to a particular topic destination, all subscribers that have subscribed to the topic automatically receive the message.

The above is referred from [TPV02]: Reference.

Our system will be using the publish/subscribe messaging system to communicate between the UOB and FDMS systems. In this way the communication model is generalized to multiple FDMS instances.



**Figure 3.5: Publish-Subscribe Messaging Model**

### 3.7.2 Enterprise Services

This includes JDBC for database access and JNDI for Java naming and directory services.

#### ◆ Java Database Connectivity (JDBC)

Java Database Connectivity (JDBC) service provides access to the databases in our systems. The JDBC API is a Java API for accessing virtually any kind of tabular data, stored in a database. The JDBC API makes it easy to send SQL statements to the database systems and supports all dialects of SQL. One can write a single program using the JDBC API to enable the program to send corresponding SQL statements to the appropriate data sources. The above is referred from [SSJ02] in the Reference Section.

JDBC services are enabled through JDBC drivers and enable the following:

- ◆ Establish a connection with a data source.
- ◆ Send queries and update statements to the data source.
- ◆ Process the results.

Applications typically make calls to the JDBC API to open a connection with a database, retrieve and update data, execute commands on the data source, and close the connections upon the end of such transactions.

As a service, the EJB container maintains a pool of database connections. This pool is transparent to the enterprise java beans. When an enterprise java bean requests a connection, the container fetches one from the pool and assigns it to the bean. This enables quick connectivity from the bean to the database. Our system will use the JDBC services on both systems to communicate with the UOB and the FDMS databases respectively.

### **Java Naming Directory Interface (JNDI)**

The Java Naming and Directory Interface (JNDI) provides naming and directory functionality to applications written using the Java programming language. These naming services provide for storage and access of various objects. They are effectively lightweight databases with very specific uses. JNDI enables applications to access resources transparently in the network. This service provides the means to locate other components in a distributed system.

Our system will use JNDI to find appropriate EJB and messaging services, for use by the various tiers.

### **3.7.3 Internet Services**

Includes support for HTTP, TCP/IP.

### **3.8 Installed Software**

Several choices of J2EE systems were available to us. They are listed as below:

- ◆ WebLogic by BEA Systems
- ◆ WebSphere by IBM

- ◆ J2EE Systems by Borland
- ◆ J2EE Systems by Oracle
- ◆ Open source systems such as Apache Tomcat, JBOSS Application Server and MySQL database server.

Each of these choices gave us a full array of J2EE capabilities. However, the open source systems offered the added advantage of offering licenses that were at no cost. Hence, we chose the following configuration (figure 3.6):

Apache Tomcat Web Server + JBOSS Application Server + MySQL Database Server

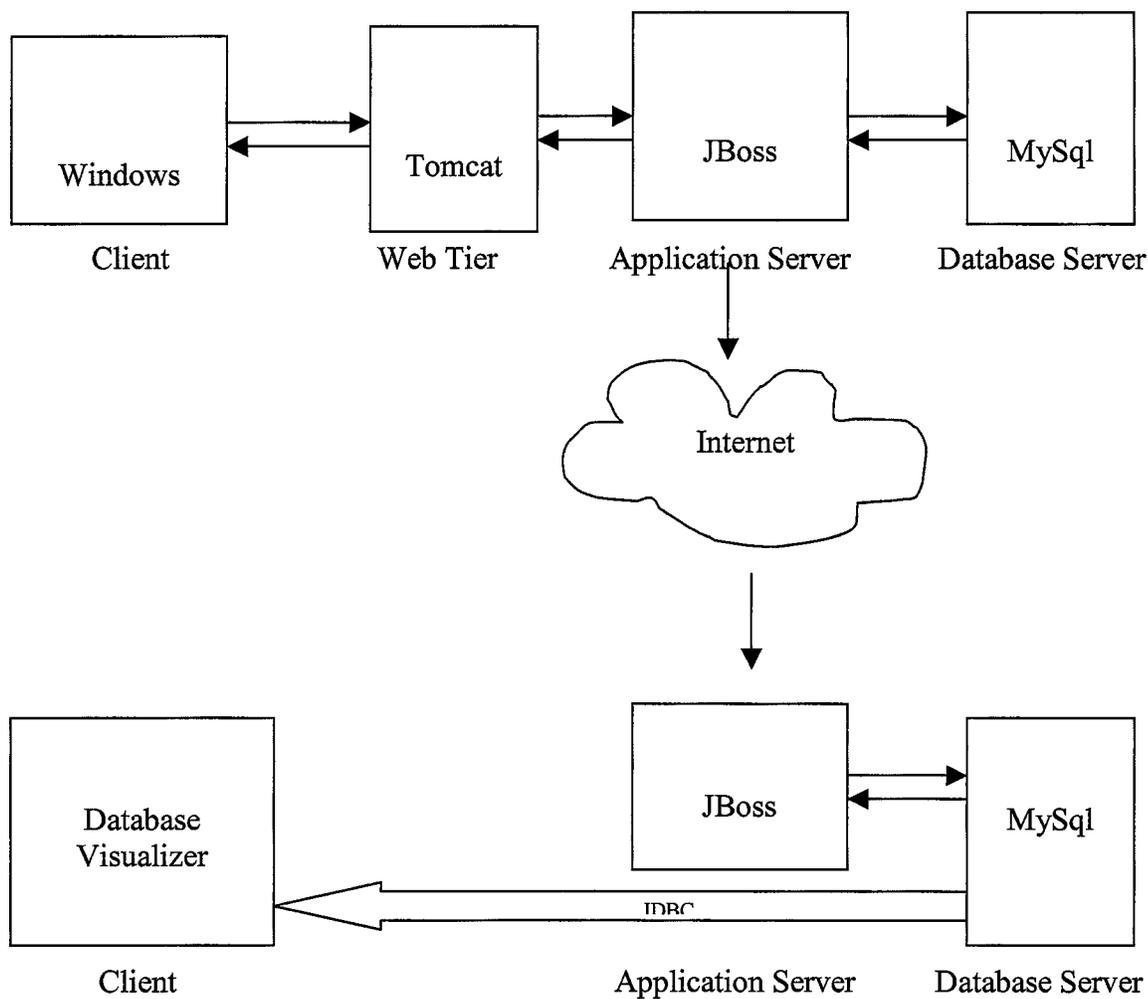
Each of these servers constitutes a tier as per J2EE nomenclature. Details of the installed software are given in the next chapter.

Our presentation, web, application and database tiers are all going to be on the same machine (one each for UOB and FDMS), but each tier is going to be on different servers.

- ◆ The presentation tier is on the browser with Windows Pro 2000 software installation.
- ◆ The web tier is on the web server, and the software installed is the Apache Tomcat version 4.0.4
- ◆ The application tier is on the application server and we have installed the Jboss server version 3.0.2.
- ◆ The database tier is on the database server, and we have installed the MySql database server version 4.0.4
- ◆ JDK (java development kit) 1.4 java, is installed on web server and application server.
- ◆ The transmission side uses Java Messaging Service for sending messages through the Publish / Subscribe Messaging Model. The transmission side publishes the message

and the receiving side subscribes to the service in order to receive the published messages.

- ◆ Web tier and application tier send messages to the databases using JDBC.
- ◆ Configured them “Eclipse Ide setup”
- ◆ Connection through JNDI.



**Figure 3.6: Installed Software**

### **3.9 Summary of the Chapter**

In this Chapter we discussed the different enabling technologies available that would meet the basic requirements of building the system and discussed the reasons for their adoption/rejection. We described in detail the enabling technology (J2EE) its components and containers, and mapped it to our system. We also showed the communication protocol that is used to communicate between tiers.

In the next chapter we will explain and illustrate in detail, the design of our system.

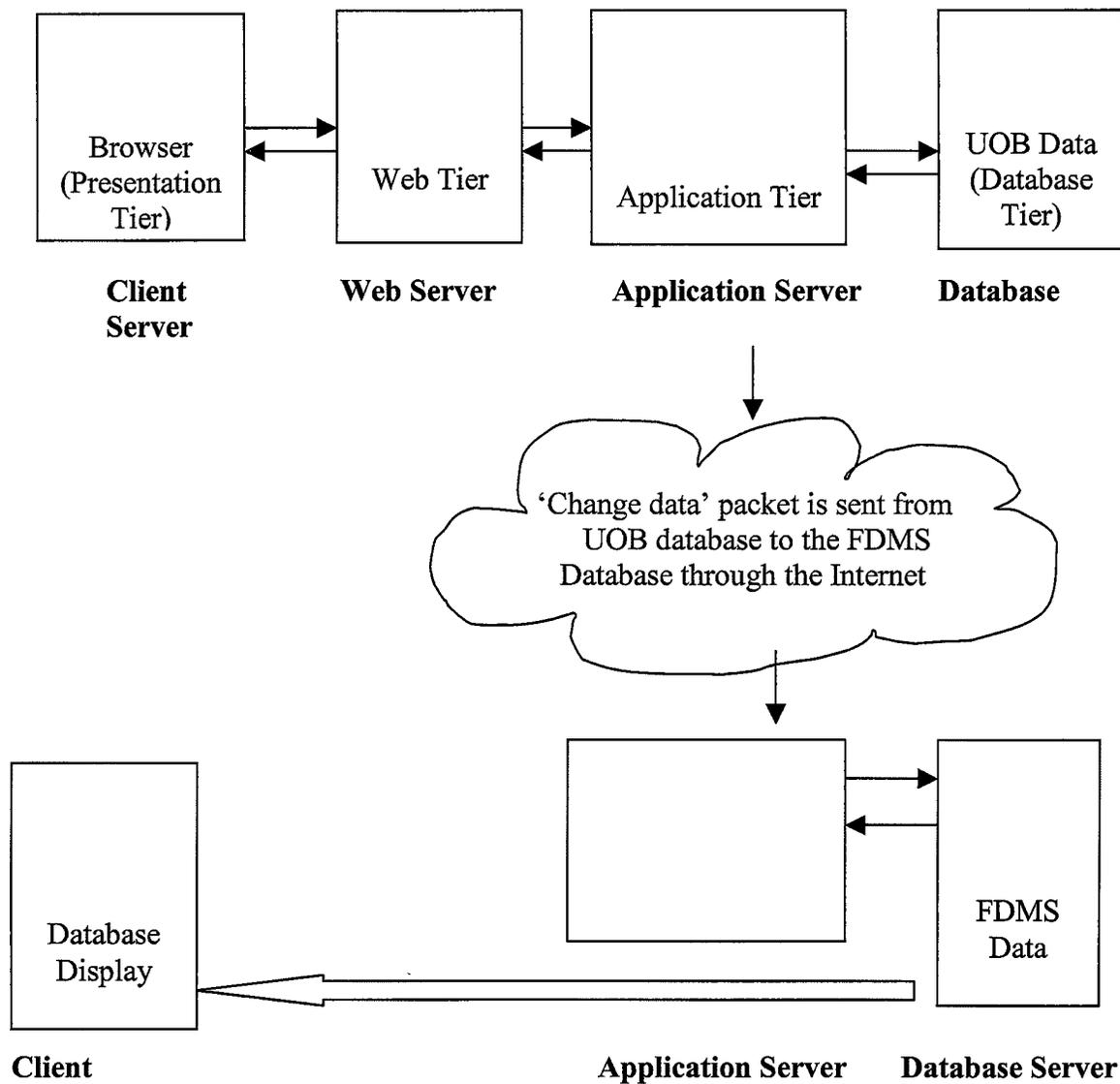
## CHAPTER 4

### SYSTEM DESIGN

#### 4.1 Introduction to the System Design

The UOB database and the FDMS database reside on different geographical locations. As discussed earlier, we have used a combination of Client Server, Web Server, Application Server and Database Server on the Transmission (UOB) Side, and a combination of Database Visualizer, Application Server and the Database Server on the Receiving (FDMS) Side. Figure 4.1 illustrates the design of the entire system.

Since the UOB and the FDMS systems reside on separate machines, we have described each side of the design separately.



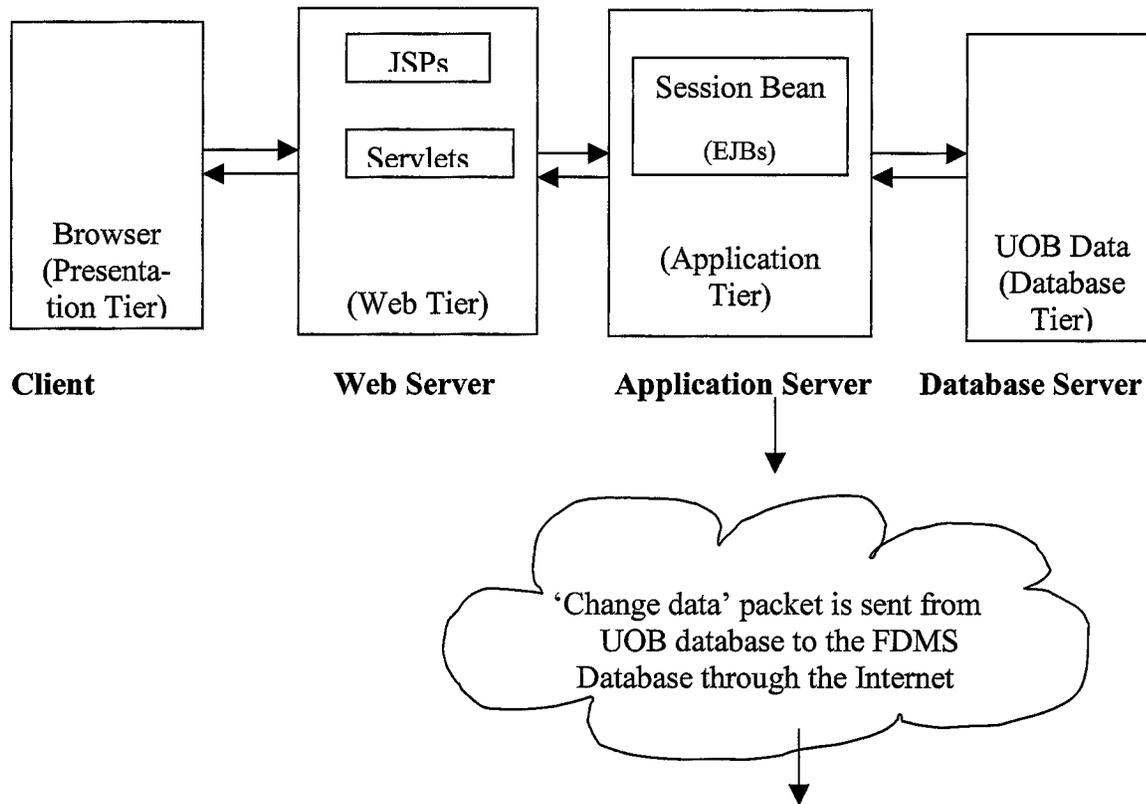
**Figure 4.1: The System Design**

## 4.2 Design of the UOB System

We will recall that the UOB Side architecture consists of:

- ◆ **The Presentation Tier** – This tier displays the browser to the user. The user will use this to communicate requests to the web tier. In our system, this could be the user requesting the display of the contents of the UOB database. The client browser manages this tier.

- ◆ **The Web Tier** – This tier will receive requests from the client browser, process them and communicate them to the application tier. Responses received from the application tier will be processed and communicated back to the presentation tier for display in the browser. Hence, for a request from the presentation tier requesting the display of the contents of the UOB database, the web tier will interpret the appropriate request, transfer it into logical service calls and make the calls on the application tier. The Web server manages this layer.
- ◆ **The Application Tier** – This tier receives requests for services from the web tier, processes them and makes corresponding requests on the database tier for data. This tier also contains the application logic for the system. In our example, this tier will receive the request for display of UOB data from the web tier and make the corresponding request on the database tier. On receipt of data from the database tier, this tier will return the data to the web tier. Additionally, the logic on this tier will also interpret requests for changes in the UOB data, made by the web tier. This tier will analyze such requests and request that corresponding changes be made in the database tier. In addition, it will check whether the changes need to be communicated to the FDMS system. If so, it will send a corresponding message to the FDMS system. The Application server manages this layer.
- ◆ **The Database Tier** – This tier manages the data in the system. It communicates with the Application Tier.



**Figure 4.2: The Transmission Side (UOB) Design**

#### 4.2.1 Design of the Presentation Tier

The presentation tier of our system is essentially a browser. Through this, the user should be able to view and modify the contents of the UOB database.

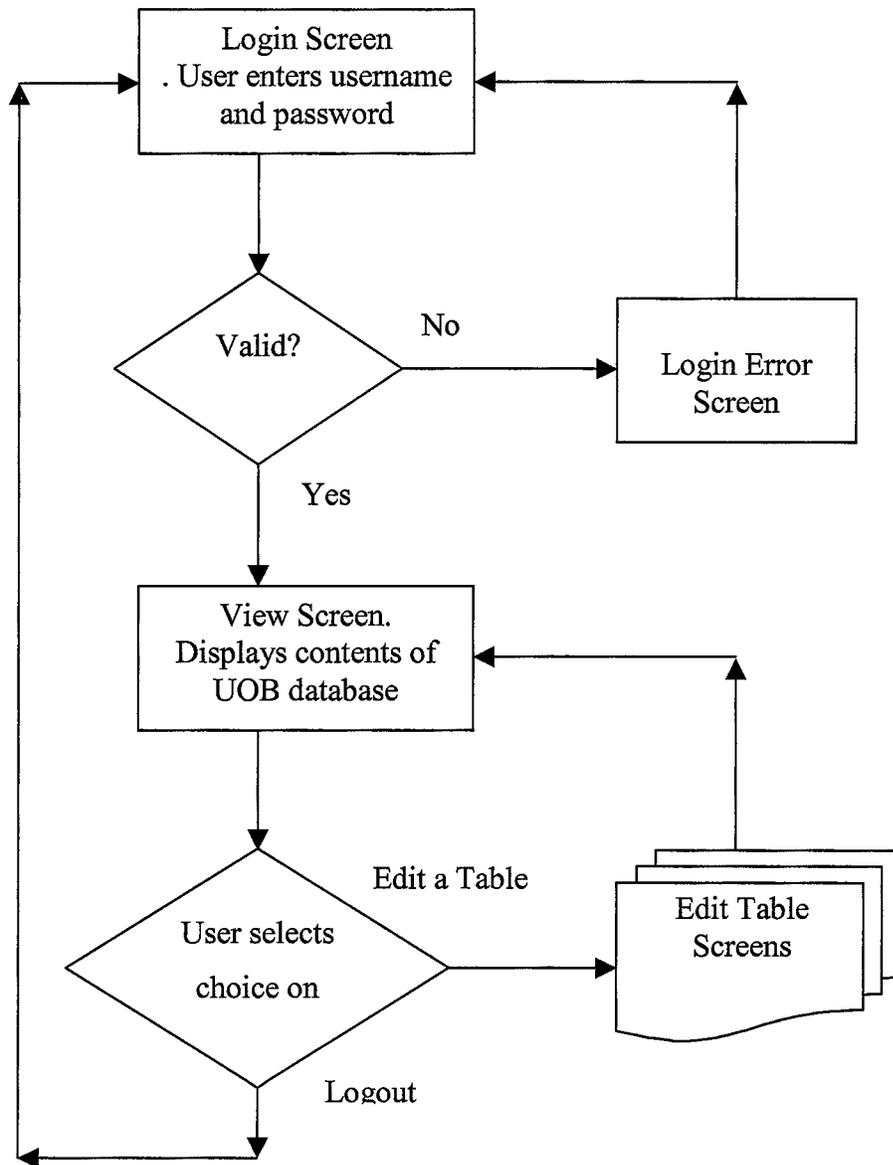
A user interface was developed utilizing separate screens for each function and for editing each table.

- ◆ **Login.** This is the first screen presented to a user, when the user accesses our system. Fundamentally, the UOB system is being designed to allow a user to change the contents of the UOB database. Since database contents need to be kept secure, we decided to create a login screen, so that only legitimate users could access it. The

login screen consists of a screen allowing a user to enter a user name and password.

If the user name and password are validated, the system will present the View screen. Else, it will present the Login Error screen.

- ◆ **Login Error.** This screen will be presented to the user when invalid user name and passwords are entered into the system. This screen will consist of an error message displayed to the user. When the user has read the message, the Login screen will be presented again.
- ◆ **View.** This screen will be presented to a user upon the successful entry of a correct set of user name and password. It will give him the ability to view the contents of the UOB database. Data modification will not be done on this screen. However, it will give the user the ability to indicate whether he would like to edit the contents of any table in the database. If a user chooses to edit any table, the appropriate Edit screen will be presented.
- ◆ **Edit.** This consists of several screens, one per table in the database. When a user indicates in the View screen that he would like to edit a table in the UOB database, the appropriate Edit screen will be displayed. In these screens, the user will be able to edit the contents of any record in the table and save it. Once saved, the system presents the View screen again.



**Figure 4.3: Pictorial Representation of Presentation Tier Design**

#### 4.2.2 Design of the Web Tier

The web tier of our system acts as the interface between the presentation tier and the application tier. All actions made by the user in the presentation tier are sent to the web tier. All screens viewed by the user in the presentation tier are created by the web tier and sent as HTML content to the browser.

When the user first opens a browser and types in the address for our system, he is essentially communicating with the web tier and requesting a login page. The web tier sends HTML content corresponding to the 'login' screen to the browser, where it is presented. When the user fills the screen with a user name and a password, the contents of the page are sent to the web tier, which in turn, sends it to the application tier for validation. If the user is validated, the web tier sends HTML content corresponding to the 'view' screen to the browser. If not, the web tier sends HTML content corresponding to the 'login error' screen to the browser.

The central program in the web tier of our application is a 'controlling servlet'. This is a program supported by the web container used in our system, Tomcat. The controlling servlet acts as the central manager of the web tier system, directing and communicating with several other servlets and Java Server Pages. The servlet is also the program receiving all communication from the browser. Based on requests received from the browser, the controlling servlet makes decisions on the next line of action. If information is to be conveyed to the application tier, it invokes an action servlet. If a screen is to be displayed, it invokes a Java Server Page.

HTML content is sent to the presentation tier using Java Server Pages (JSPs), as defined in Chapter 3. Java Server Pages are specialized pages supported by the web container for

the dispatch of HTML content to the browser. Each screen (Figure 4.26) has its own corresponding JSP in the web tier. When it's time to present a certain screen to a user, the controlling servlet invokes the corresponding JSP. The JSP creates appropriate HTML formatting code for the screen and sends it to the browser. If data from the system is to be presented on the screen, the JSP obtains it from the application tier using appropriate service calls. Each JSP can communicate directly with the application tier to obtain data for presentation.

The web tier also uses other programs called 'action servlets' for communicating with the application tier. These are programs that receive information from the controlling servlet and make corresponding service calls on the application tier.

The figures in table 4.1 and 4.2 show the sequence of actions between the presentation tier and the web tier, detailing the interaction between the various components of the web tier.

Presentation Tier (Contents of browser)	User Action	Web Tier
Blank on startup	User types in the URL of our system into a browser	Controlling servlet receives communication for startup screen. Invokes the 'Login' JSP. Login JSP sends HTML for login screen to browser
Login Screen	User types in <b>incorrect</b> username and password and clicks on the 'Login' button	Controlling servlet receives username and password for validation. Invokes action servlet to send it to application tier. Waits for validation from application tier. Since this was not validated (incorrect password), invokes 'Login Error JSP'. Login Error JSP sends HTML for login error screen to the browser
Login Error Screen	User clicks on the OK button signaling user has read error message and is ready to enter username and password again	Controlling servlet invokes 'Login' JSP. Login JSP sends HTML for login screen to browser

**Table 4.1: Tabular Representation of Web Tier Design (continued in 4.1(a))**

Continued from 4.1:

Presentation Tier (Contents of browser)	User Action	Web Tier
Login Screen	User types in <b>correct</b> username and password and clicks on the 'Login' button	Controlling servlet receives username and password for validation. Invokes action servlet to send it to application tier. Waits for validation from application tier. Since this is validated (correct password), invokes 'View JSP'. View JSP requests UOB data from the application tier, formats it into HTML and sends it to the browser to display contents of UOB database in read only format
View Screen	User selects option to edit the Unit table and clicks on the OK button  (Editing of other tables is identical)	Controlling servlet receives user option. Invokes the 'EditUnit Table JSP'. EditUnitTable JSP requests contents of the Unit table of the UOB database from the application tier. Upon receiving it, it formats the data into an HTML table and sends it to the browser to display in editable format
Edit Unit Table Screen	User makes changes to some of the values in the Unit table and clicks on 'save'	Controlling servlet receives edited values for the Unit table and calls on the action servlet to request the application tier to save it in the UOB database. When done, it invokes the View JSP to display the updated values on the screen.
View Screen	User selects 'Logout'	Controlling servlet invokes the Login JSP.
Login Screen		

Table 4.1 (a): Tabular Representation of Web Tier Design

### 4.2.3 Design of the Application Tier

The application tier of our system on the UOB side consists of application logic, embedded in a stateless session enterprise java bean (EJB). Stateless session EJBs are defined in Chapter 3. These EJBs are supported by the application server used in our system, JBoss.

The application server provides essential communication services to the EJB, allowing the web tier to communicate easily with the EJB, through programming calls. The EJB exposes a series of application programming interface (API) calls, used by the web tier to request services from the application tier. The application server also provides the EJB with Java Messaging capability (JMS), allowing the EJB to send packets of 'change' data to the FDMS system.

The EJB communicates with the database using another protocol, called Java Data Base Connectivity (JDBC), also defined in Chapter 3. JDBC allows SQL statements to be embedded in programming calls, allowing the EJB to interact with the UOB database.

The EJB contains all the logic needed for the processing of services requested by the web tier. It also communicates with the UOB database for update and retrieval of data.

Additionally, it contains a map of common data between the UOB and FDMS databases, allowing it to send packets of 'update' information to the FDMS system whenever data common to both databases is edited.

The table in Figure 4.2 details the services requested by the web tier from the application tier and the corresponding design of the application tier.

<b>Service requested by Web Tier</b>	<b>Data Interaction In / Out</b>	<b>Application Tier Functionality</b>
User Validation	User name and password	Accepts a set of user name and password and tries to match with existing accounts in the database. Returns True if a match is found; False otherwise
	True / False	
Retrieve all UOB data	None	Retrieves all UOB data in the database and returns to the web tier
	UOB data	
Retrieve UOB data in a specific table	UOB table name	Retrieves all UOB data in specified table from the database and returns it to the web tier
	UOB data in specified table	
Update UOB data in a specific table	UOB table name, updated data	<ul style="list-style-type: none"> <li>• Updates the specified table in the UOB database with the supplied data.</li> <li>• Checks mapping table to see whether this was common data with FDMS</li> <li>• Creates a packet of 'change' data, if true</li> <li>• Sends 'change' packet to FDMS system</li> </ul>
	None	
Others	None	Performs initialization of connection with the database
	None	

**Table 4.2: Tabular Representation of Application Tier Activities**

#### 4.2.4 Design of the Database Tier

The database tier consists of a database. In our system, we used the MySQL database.

The design of the database was the simplest part of the entire system. It was done using the SQL language, from a command line window.

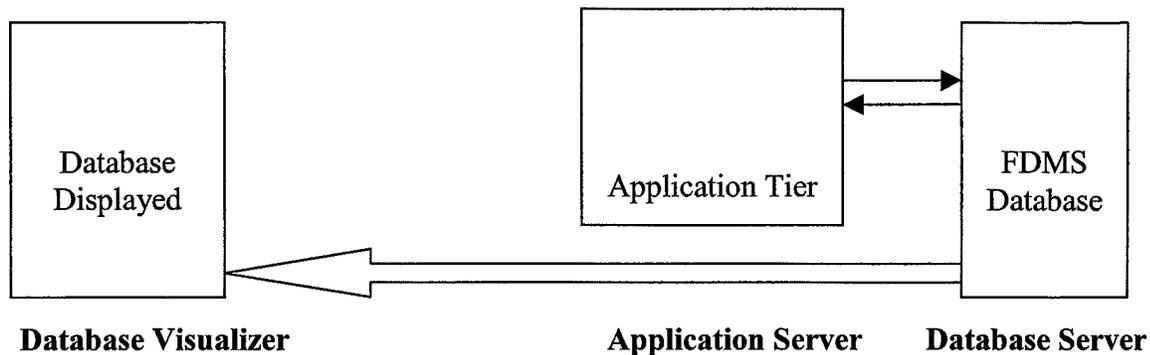
The design consisted of the following steps:

- ◆ Create a database for our use in the MySQL database
- ◆ Create user permissions and privileges for our system to access the database
- ◆ Create tables corresponding to the UOB database.

#### 4.3 Design of the Receiving Side (FDMS) System

The receiving side system receives change packets from the UOB system, decodes them and updates the FDMS database (figure 4.4). It consists of:

- ◆ **Database Visualizer** - Since data will be updated in the database, we did need the ability to view its contents. However, we decided not to create a presentation tier for it since a public domain application was available for that purpose. Hence, in our system, data in the FDMS database will be viewed using an external application, called the Database Visualizer.
- ◆ **Application tier** – This tier contains all of the logic for the messaging interface, decoding of the change packets and updating of the database; and the FDMS database tier.
- ◆ **Database Tier** – This tier manages the FDMS-side data.



**Figure 4.4: Transmission Side (FDMS) Design**

### 4.3.1 Design of the Database Visualizer

The Database Visualizer is an application written by Minq Software, Sweden. It allows the visualization of contents of a database. We used version 3.3.1, since it met our needs and was freely downloadable.

The application was downloaded from the web site [www.minq.se/products/dbvis/install-331/install.jsp](http://www.minq.se/products/dbvis/install-331/install.jsp). The downloaded file was an executable file, which when executed, installed the application on our system.

The application was configured as follows:

1. We had to give it the path of the JDBC driver for MySQL, installed on our system. Our driver was in the file “com.mysql.jdbc.driver” and was obtained along with the MySQL database.
2. We had to give it the URL of the MySQL system whose tables we wanted to view. In our case, it was ‘localhost’.
3. We had to give it a valid user name and password to allow it to access the tables of our database.

When the application was configured with these details, we were able to view all the tables of our database.

### **4.3.2 Design of the Application Tier**

The application tier on the FDMS side consists of two fundamental elements, both supported by the application server. The first one is the Message Driven Bean (MDB), introduced in Chapter 3. The second one is a message queue.

The message driven bean is a special EJB supported by the application server. Like other EJBs it contains program logic and, in our context, has all the ability to make JDBC calls and interface with the database. Unlike other EJBs however, it does not expose an application-programming interface. It is not called externally by any other entity or component.

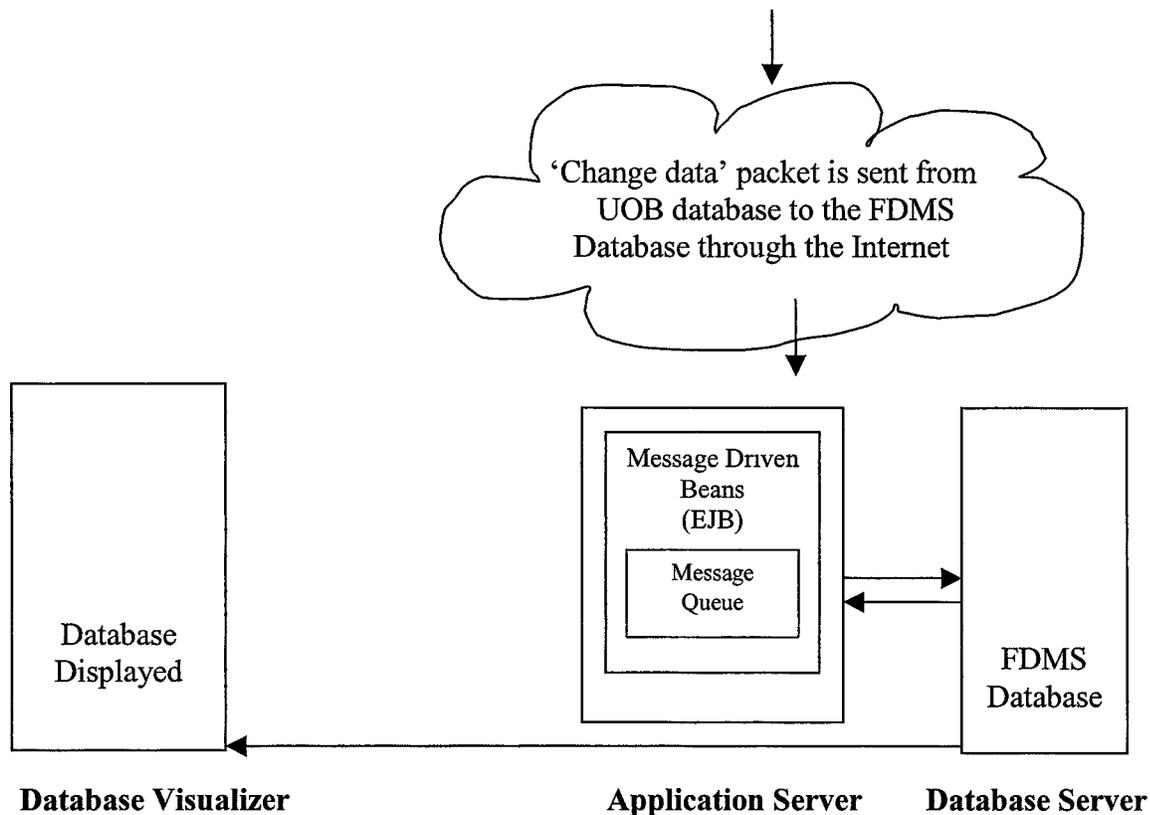
Each MDB is a message listener. It (indirectly) listens to a topic or a queue. Unless a message arrives on that topic or in the queue, it is not invoked. When a message it's listening on arrives, the application server invokes it and calls its 'OnMessage () method. That allows an MDB to execute in that context. Hence, unlike other EJBs, which support synchronous operations, an MDB fundamentally supports asynchronous communication between components of a system.

Our FDMS system uses a message queue for communication. JBoss supports the creation of JMS message queues and allows components to send messages to that queue by using its address. The UOB system sends change packets in form of JMS messages to this queue. Our MDB is programmed to be a listener on this queue.

When a 'change' packet arrives from the UOB system, the application server invokes the MDB and calls its OnMessage() method, passing the 'change' packet to it as a parameter. Once invoked, the MDB functions similar to a stateless session EJB. It decodes the 'change' packet and extracts the information needed to update the necessary tables and records in the FDMS database. It then uses the JDBC interface to update the database, in effect synchronizing the databases.

Hence, the design of the application tier is summarized as follows:

- ◆ Create a JMS message queue
- ◆ Create an MDB as a listener to this queue
- ◆ When a message arrives at this queue, the application server invokes the MDB and sends the message to it
- ◆ The MDB decodes the message and extracts the 'change' information out of it.
- ◆ The MDB updates the FDMS database as per the 'change' information.



**Figure 4.5: Transmission Side (FDMS) Design**

### 4.3.3 Design of the Database Tier

The database tier consists of the FDMS database. In our system, we used the MySQL database. The design of the database was done using the SQL language, from a command line window.

The design consisted of the following steps:

- ◆ Create a database for our use in the MySQL database. The name of our database was 'soup'.
- ◆ Create user permissions and privileges for our system to access the database.

#### 4.4 The Complete System Architecture

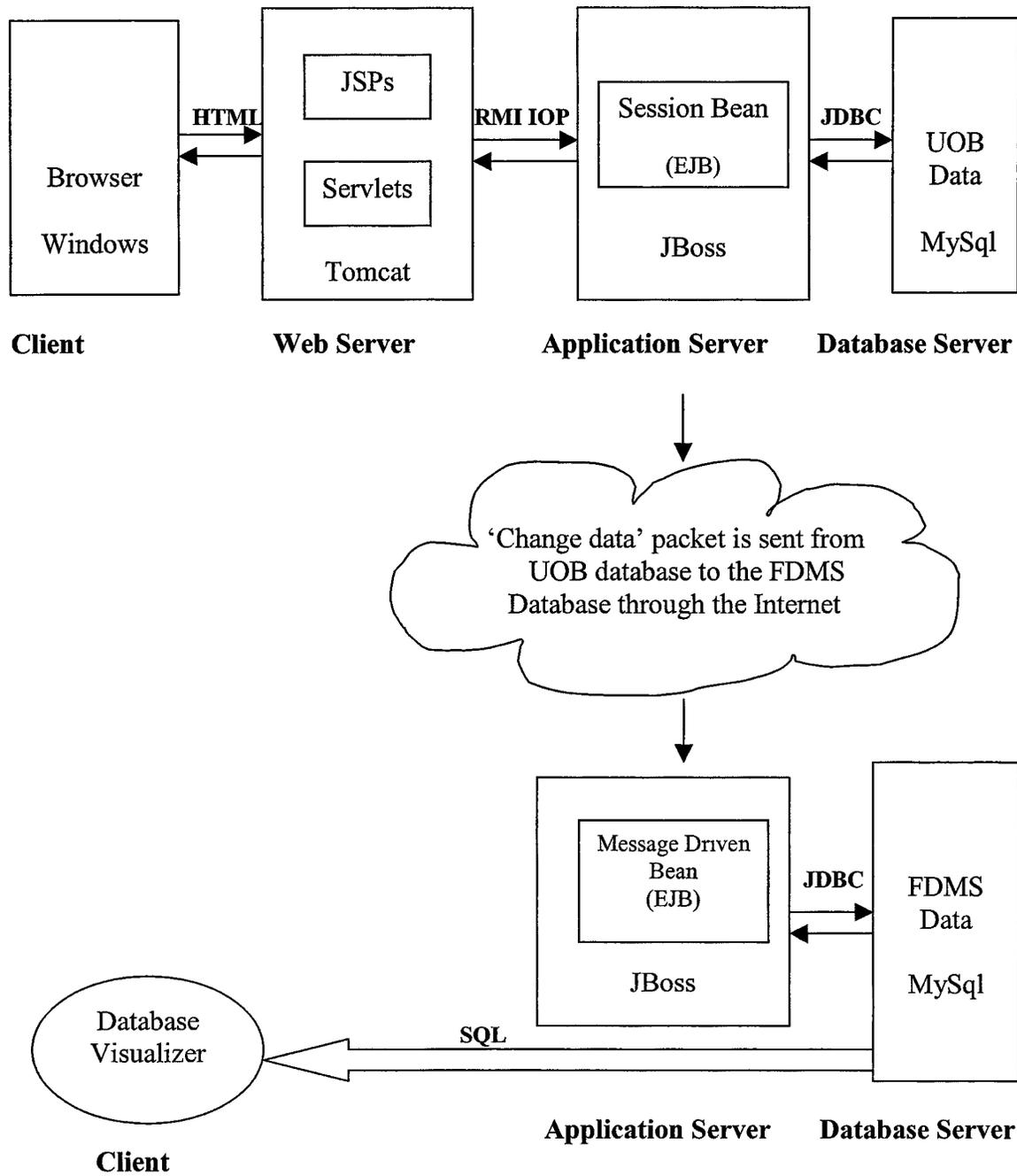


Figure 4.6: Systems Architecture

- ◆ Create tables corresponding to the FDMS database. The tables created were ‘Entity’, ‘Entity Component’, ‘Characteristic’ and ‘Entity Characteristic’.

#### **4.5 Summary of the Chapter**

In this chapter we discussed the design of our system. We displayed and explained the systems different components and their functionalities and the combination of software used to construct the system.

In the next chapter we will describe the detailed high and low-level implementation process.

## **CHAPTER 5**

### **PROJECT IMPLEMENTATION**

#### **5.1 Introduction to implementation**

In the preceding chapters we have described the design of our system and the technology we will be using to create it. The system as described, is made of the Transmission Side (UOB), and the Receiving Side (FDMS). In this chapter we will explain user actions and corresponding system implementation.

#### **5.2 Transmission Side**

##### **5.2.1 Implementation Description of Transmission Side Modules**

###### **5.2.1.1 User Login**

###### **A. High Level Design**

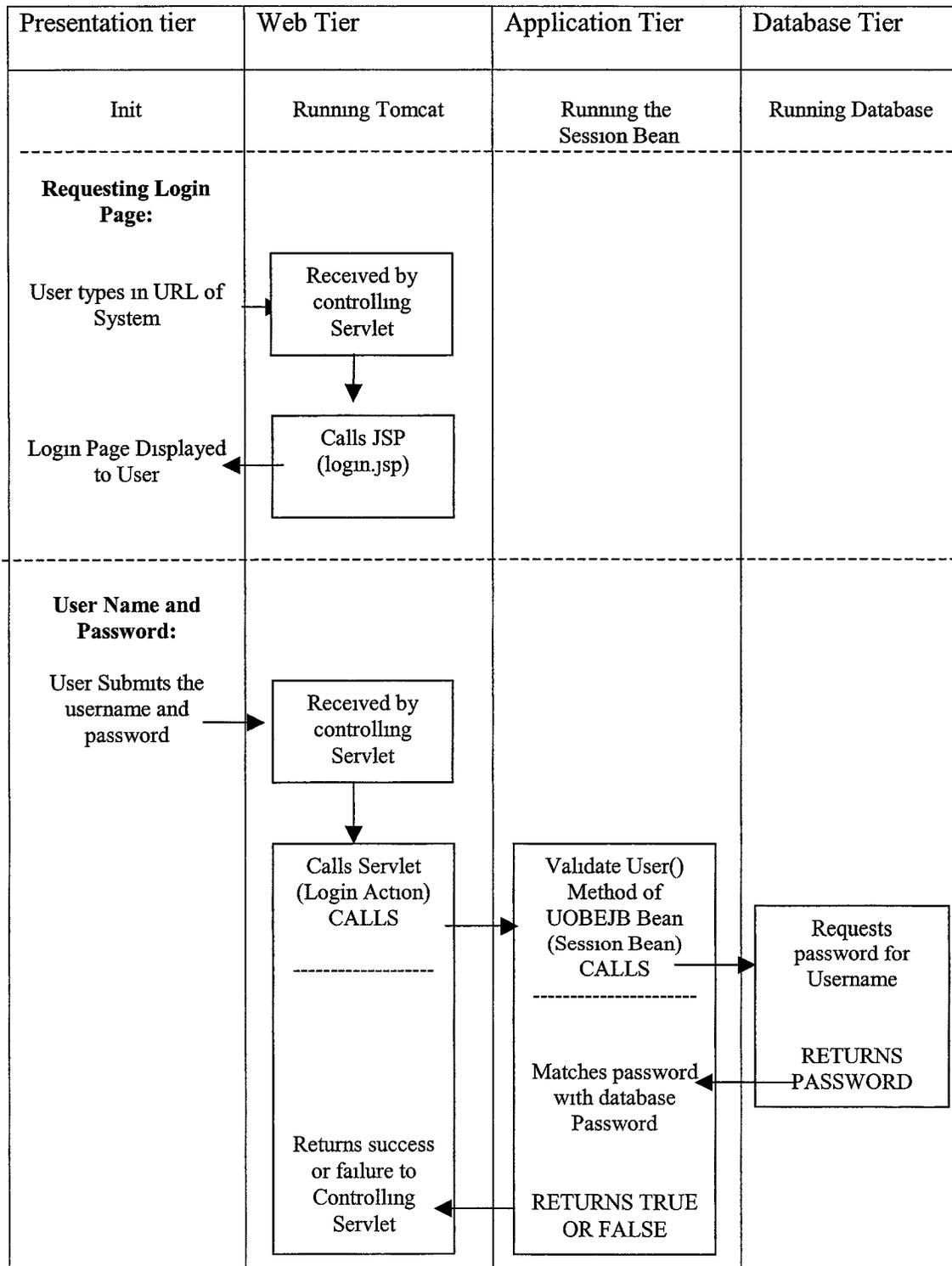
User accesses the website, and logs in with his User Name and Password to access the UOB data (figure 5.1). This fulfils the requirement specification in chapter 2, figure 2.3.

**UOB Data Editor: Login**

User Name	<input type="text"/>
Password	<input type="password"/>

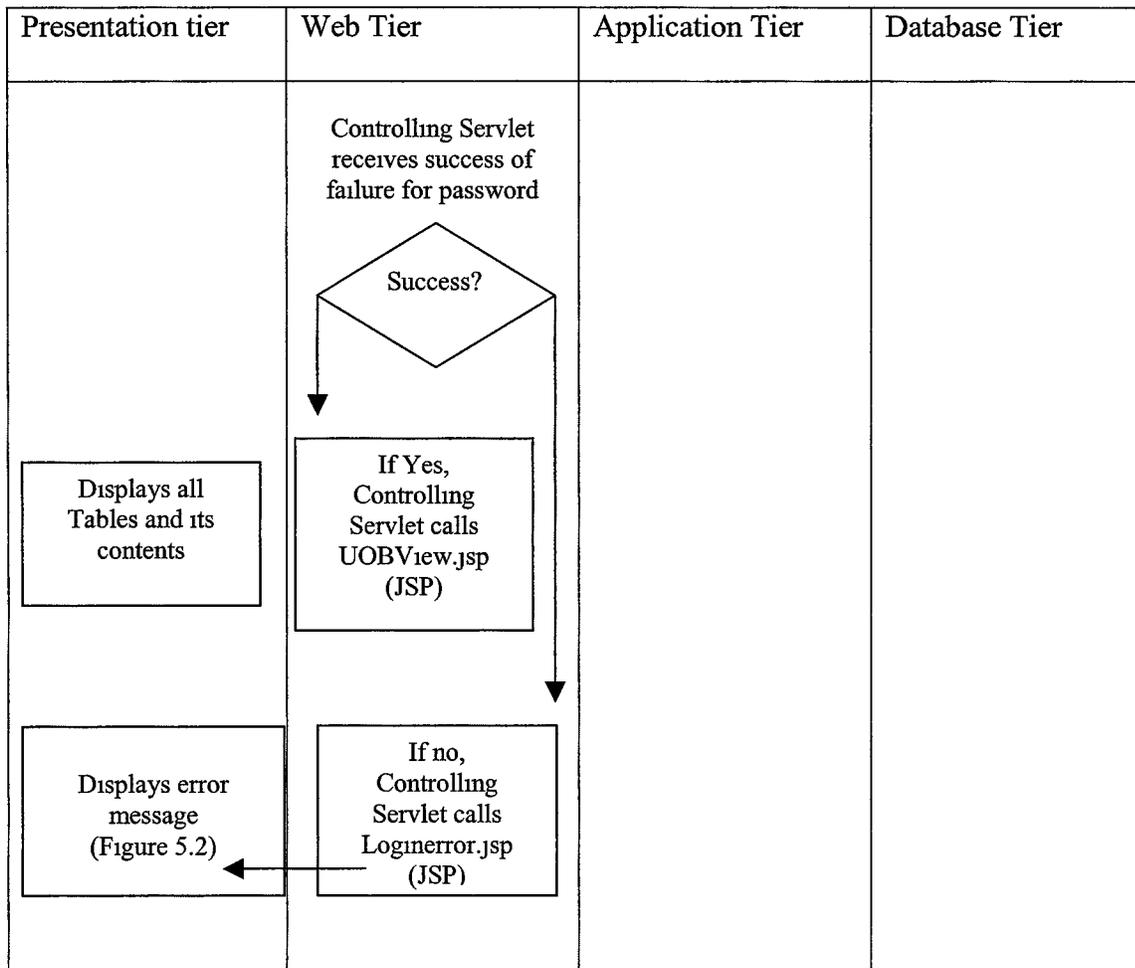
**Figure 5.1: User Login**

## B. Low Level Design



**Table 5.1: User Login (Continued on 5.1(a))**

Continued from 5.1:



**Table 5.1 (a): User Login**

- ◆ When the user submits the Username and Password data on the ‘Login’ screen, the data is sent to the Controlling Servlet in the Web Tier.
- ◆ The Controlling Servlet accepts the Username and Password and calls the Login Action servlet to validate them.
- ◆ The Login Action Servlet calls the ‘ValidateUser’ method in the UOB EJB, in the Application Tier to validate them.
- ◆ The UOB EJB searches the database for a password related to the given Username. If the password matches correctly, it returns ‘true’ to the Login Action servlet. Else, if

either the Username is not found in the database or the password does not match, it returns 'false'.

- ◆ The Login Action servlet, based on the results received, returns 'success' or 'failure' to the controlling servlet.
- ◆ If the result was a failure, the controlling servlet invokes 'ErrorLogin.jsp', which displays the 'Login Error' html page (Figure 5.2). It communicates to the user that the log in attempt failed. When the user clicks 'OK' on this page, the controlling servlet invokes 'Login.jsp' again to allow the user to re-enter the user name and password.

**UOB Data Editor: Login Error!!**

**Login activity had error(s):**

**Please try again...**



**Figure 5.2: Login Error**

If the result is a success, the Controlling Servlet invokes the JSP 'UobView.jsp'.

- ◆ 'UobView.jsp' represents the UOB Editor page and has to display the contents of all the tables in the database (figure 5.3). For each table to be displayed, it calls the 'ViewObjects' method in the UOB EJB to get the contents of that table.
- ◆ The ViewObjects method in the EJB makes a call to the database requesting all the contents for that specific table. Upon receipt, it returns them to UobView.jsp.

- ◆ The UobView.jsp displays the contents of that table in the UOB database. It repeats this sequence for all the tables in the database, displaying them all. It also provides the user several buttons by which he can either modify (Create / Delete / Edit) the contents of any table or log out (Logout) of the application. (Figure 5.2)

### Welcome to UOB Editor

#### Table View

[Logout](#)

**Unit**

Unit Identification Code	Parent Unit Id. Code	Unit Name	Home Name	Ship Category	Country Code	Select
ARMR BN	null	Armor Battalion	Pink	N1	US	<input type="checkbox"/>
BN HQ	ARMR BN	Armor Battalion Headquarters	Red	N2	US	<input type="checkbox"/>
ARMR CO	ARMR BN	Armor Company	Brown	N3	US	<input type="checkbox"/>
CO HQ	ARMR CO	Armor Company Headquarters	White	N4	US	<input type="checkbox"/>
ARMR PLT	ARMR CO	Armor Platoon	Orange	N5	US	<input type="checkbox"/>
CMD SN	ARMR PLT	Armor Command Section	Green	N6	US	<input type="checkbox"/>
ARMR SN	ARMR PLT	Armor Section	Yellow	N7	US	<input type="checkbox"/>

**Table 5.2: UOB Database Tables (Continued in Table 5.2 (a))**

(Continued from Tables 5.2):

**Unit-Aircraft**

<b>AIC</b>	<b>Unit Identification Code</b>	<b>Aircraft Code</b>	<b>Aircraft Description</b>	<b>Aircraft Qty Required</b>	<b>Aircraft Qty Auth</b>	<b>Select</b>
UH1V	BN HQ	H13579	UH-1V Utility Helicopter	2	2	<input type="radio"/>
UH1V	CO HQ	H13589	UH-1V Utility Helicopter	4	4	<input type="radio"/>

**Table 5.2(a): UOB Database Tables (Continued in Table 5.2 (b))**

Continued from Tables 5.2 (a):

<b>Unit-Equipment</b>						
		<a href="#">Create</a>	<a href="#">Delete</a>	<a href="#">Edit</a>		
<b>EIC</b>	<b>Unit Identification Code</b>	<b>Equipment Code</b>	<b>Equipment Description</b>	<b>Equipment Qty Required</b>	<b>Equipment Qty Auth</b>	<b>Select</b>
M1A1	BN HQ	M1234	Main Battle Tank	15	11	<input type="radio"/>
M23	BN HQ	M2345	7.62 MM Sidearm	145	145	<input type="radio"/>
M1R	BN HQ	M1357	Rifle	61	61	<input type="radio"/>
M1A1	CO HQ	M1234	Main Battle Tank	15	11	<input type="radio"/>
M23	CO HQ	M2345	7.62 MM Sidearm	145	145	<input type="radio"/>
M1R	CO HQ	M1357	Rifle	61	61	<input type="radio"/>
M1A1	CMD SN	M1234	Main Battle Tank	15	11	<input type="radio"/>
M23	CMD SN	M2345	7.62 MM Sidearm	145	145	<input type="radio"/>
M1R	CMD SN	M1357	Rifle	61	61	<input type="radio"/>
M1A1	ARMR SN	M1234	Main Battle Tank	15	11	<input type="radio"/>
M23	ARMR SN	M2345	7.62 MM Sidearm	145	145	<input type="radio"/>
M1R	ARMR SN	M1357	Rifle	61	61	<input type="radio"/>

**Table 5.2 (b): UOB Database Tables (Continued in Table 5.2 (c))**

Continued from Tables 5.2 (b):

Unit-Personnel	<input type="button" value="Create"/> <input type="button" value="Delete"/> <input type="button" value="Edit"/>				
	PIC	Unit Identification Code	Personnel Description	Personnel Qty Required	Personnel Qty Auth
LT COL	BN HQ	Lieutenant Colonel	1	1	<input type="radio"/>
MAJ	BN HQ	Major	3	2	<input type="radio"/>
DRIVER	BN HQ	Tank Driver	4	4	<input type="radio"/>
GUNNER	BN HQ	Gunner	6	5	<input type="radio"/>
LOADER	BN HQ	Loader	6	5	<input type="radio"/>
PILOT	BN HQ	UH-1 Pilot	2	2	<input type="radio"/>
CAPT	CO HQ	Captain	4	4	<input type="radio"/>
1LT	CO HQ	First Lieutenant	5	4	<input type="radio"/>
DRIVER	CO HQ	Tank Driver	4	4	<input type="radio"/>
GUNNER	CO HQ	Gunner	6	5	<input type="radio"/>
LOADER	CO HQ	Loader	6	5	<input type="radio"/>
PILOT	CO HQ	UH-1 Pilot	2	2	<input type="radio"/>
2LT	CMD SN	Second Lieutenant	5	4	<input type="radio"/>
DRIVER	CMD SN	Tank Driver	4	4	<input type="radio"/>
GUNNER	CMD SN	Gunner	6	5	<input type="radio"/>
LOADER	CMD SN	Loader	6	5	<input type="radio"/>
CHIEF	ARMR SN	Crew Chief	2	2	<input type="radio"/>
DRIVER	ARMR SN	Tank Driver	4	4	<input type="radio"/>
GUNNER	ARMR SN	Gunner	6	5	<input type="radio"/>
LOADER	ARMR SN	Loader	6	5	<input type="radio"/>

Table 5.2 (c): UOB Database Tables

### **5.2.1.2 Updating Database Tables**

The UOB Editor allows a user to access all the UOB tables in the database (figure 5.3).

For each table, the user can click on the

1. 'Create' button for adding a new row (entry) in a table.
2. 'Delete' button for deleting an existing row.
3. 'Edit' button for modifying a specific row of a table.

The Unit table allows no editing. Additionally, the user can click on the Logout button to exit the application. We will be describing the activities for one table (Unit Aircraft). The implementation for all tables is identical.

#### **1. Creating a New Row in a Table**

##### **◆ Accessing the Data**

##### **A. High Level**

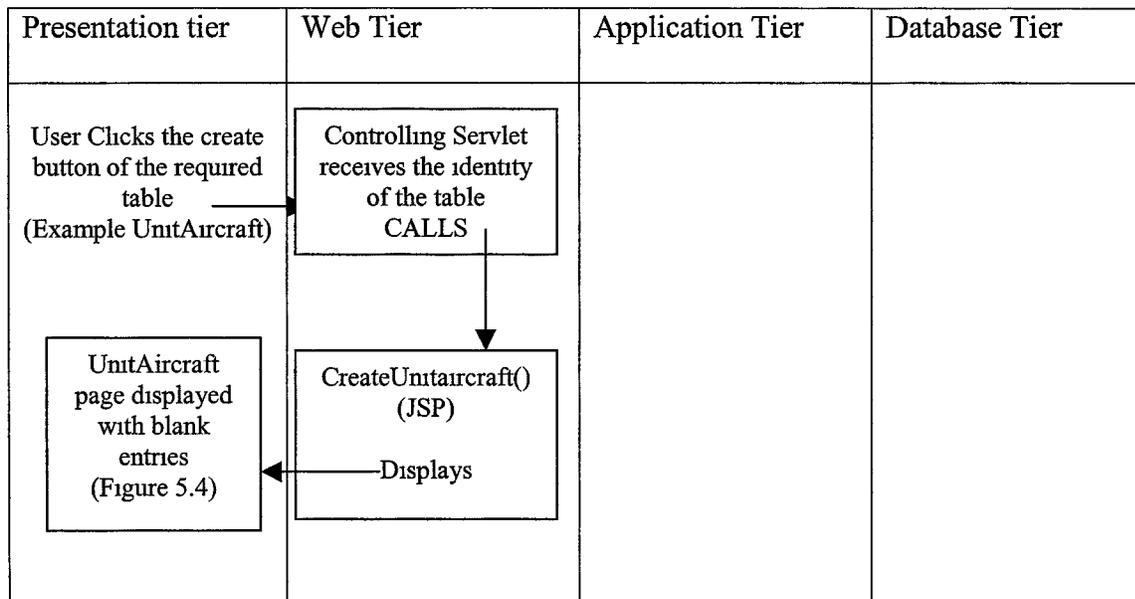
The user clicks on the 'Create' button of the Unit Aircraft table (figure 5.3) and the page 'Create Unit Aircraft' is displayed to him (figure 5.4).

**UOB Editor****Create Unit Aircraft****Unit Aircraft**

<input type="text"/>	AIC
<input type="text"/>	Unit Id
<input type="text"/>	Aircraft Code
<input type="text"/>	Aircraft Desc
<input type="text"/>	Aircraft Qty Reqd
<input type="text"/>	Aircraft Qty Auth

**Figure 5.4: Creating a New Unit Aircraft Table**

## B. Low Level



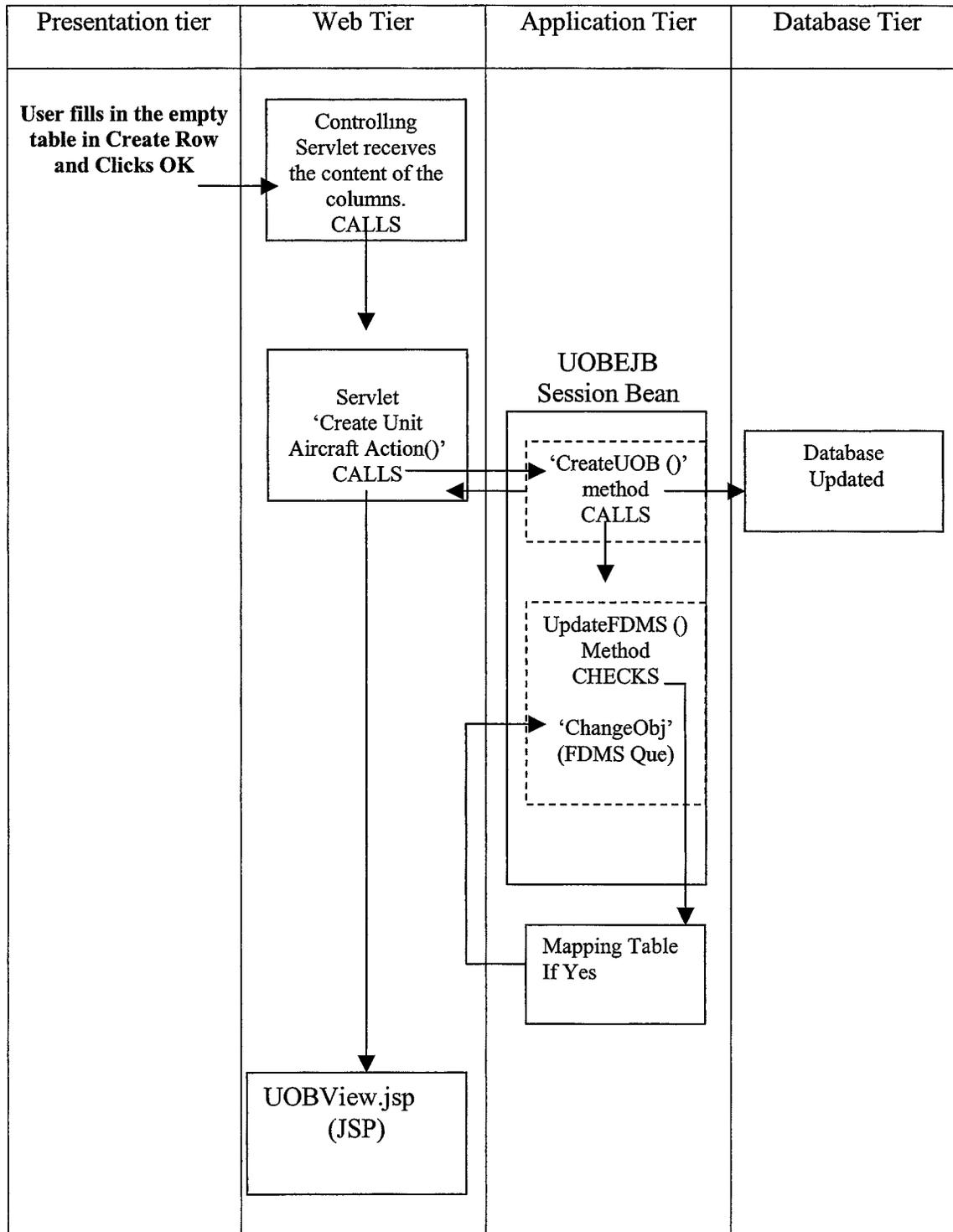
**Table 5.3: Accessing Table to Create Row**

- ◆ The Controlling Servlet (in the Web Tier) receives the identity of the table for which the user has requested to add a new row – in this case the Unit Aircraft table.
- ◆ The Controlling Servlet invokes the JSP ‘CreateUnitAircraft.jsp’.
- ◆ The CreateUnitAircraft.jsp displays the ‘Create Unit Aircraft’ html page (Figure 5.4). This presents to the user a page with blank entries, corresponding to the columns in a row of the Unit Aircraft table.
- ◆ **Entering data in the row**

## A. High Level

The user fills in the columns and clicks the ‘OK’ button.

**Low Level**



**Table 5.4: Creating a New Row**

- ◆ The Controlling Servlet receives the contents of the columns the user just filled in.
- ◆ It calls the 'Create Unit Aircraft Action' servlet and passes to it all the data received from the user entry.
- ◆ The 'Create Unit Aircraft Action' servlet calls the 'CreateUOB' method of the UOBEJB bean in the Application Tier and passes to it the identity of the table (Unit Aircraft) and all the data for inserting into a new row in this table.
- ◆ The CreateUOB method sends the data to the database for insertion into a new row in the Unit Aircraft table.
- ◆ The CreateUOB method then calls the UpdateFDMS method in the UOBEJB to update the FDMS tables, if required.
- ◆ The UpdateFDMS method checks in a mapping table to see if the data is mapped into the FDMS database. If so, it creates a 'ChangeObj' object and inserts into it the identity of the operation ('create'), the identity of the table (Unit Aircraft), and the new data itself. Then, it pushes the 'ChangeObj' object to the queue of the FDMS system. When done, it returns control to the CreateUOB method, which in turn, indicates to the Create Unit Aircraft Action servlet that it's done.
- ◆ The 'Create Unit Aircraft Action' servlet indicates to the Controlling Servlet that it is done.
- ◆ The Controlling Servlet calls UobView.jsp. UobView.jsp gets all the current (including the new row added) contents of the database and displays them to the user in the UOB Editor html page. At this point, the system waits for new user input.

## 2. Deleting a Row in a Table

### A. High Level

The user selects a row in the Unit Aircraft table and clicks on the 'Delete' button of the Unit Aircraft table (figure 5.3).

- ◆ The Controlling Servlet (in the Web Tier) receives the identity of the table (Unit Aircraft) and the number of the row that the user has requested to delete.
- ◆ The Controlling Servlet calls the 'Delete Action' servlet and passes to it the table name and row number.
- ◆ The 'Delete Action' servlet calls the 'DeleteUOB' method of the UOBEJB bean in the Application Tier and passes to it the identity of the table (Unit Aircraft) and the identity of the row to be deleted.
- ◆ The DeleteUOB method sends the data to the database requesting deletion of the row in the Unit Aircraft table.
- ◆ The DeleteUOB method then calls the UpdateFDMS method in the UOBEJB to update the FDMS tables, if required.
- ◆ The UpdateFDMS method checks the mapping table to see if the data is mapped into the FDMS database. If so, it creates a 'ChangeObj' object and inserts into it the identity of the operation ('delete'), the identity of the table (Unit Aircraft), and keys for identification of the row that was deleted. Then, it pushes the 'ChangeObj' object to the queue of the FDMS system. When done, it returns control to the DeleteUOB method, which in turn, indicates to the Delete Action servlet that it's done.
- ◆ The 'Delete Action' servlet indicates to the Controlling Servlet that it is done.

Low Level

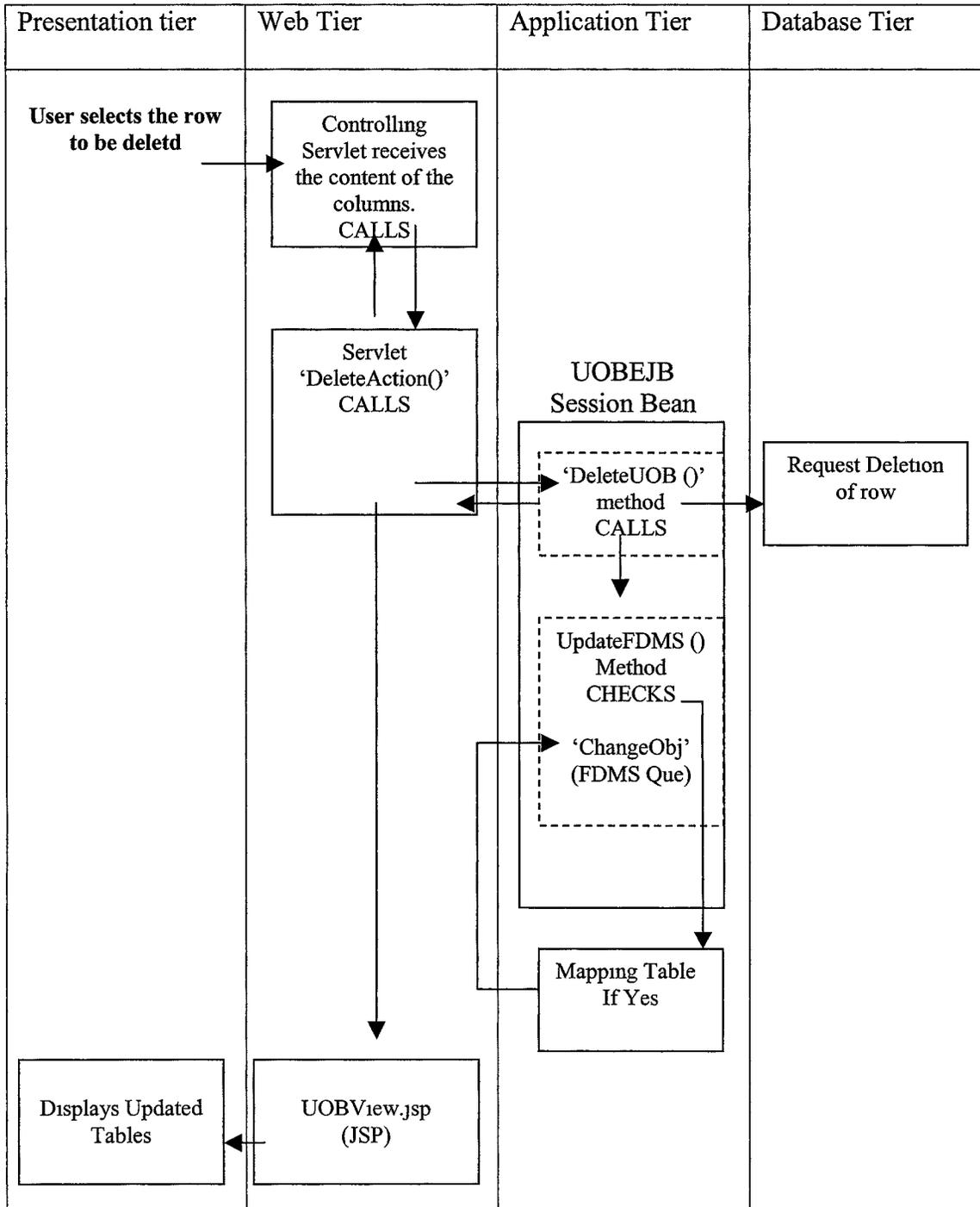


Table 5.4: Deleting a Row

- ◆ The Controlling Servlet calls UobView.jsp. UobView.jsp gets all the current (new) contents of the database and displays them to the user in the UOB Editor html page.

At this point, the system waits for new user input.

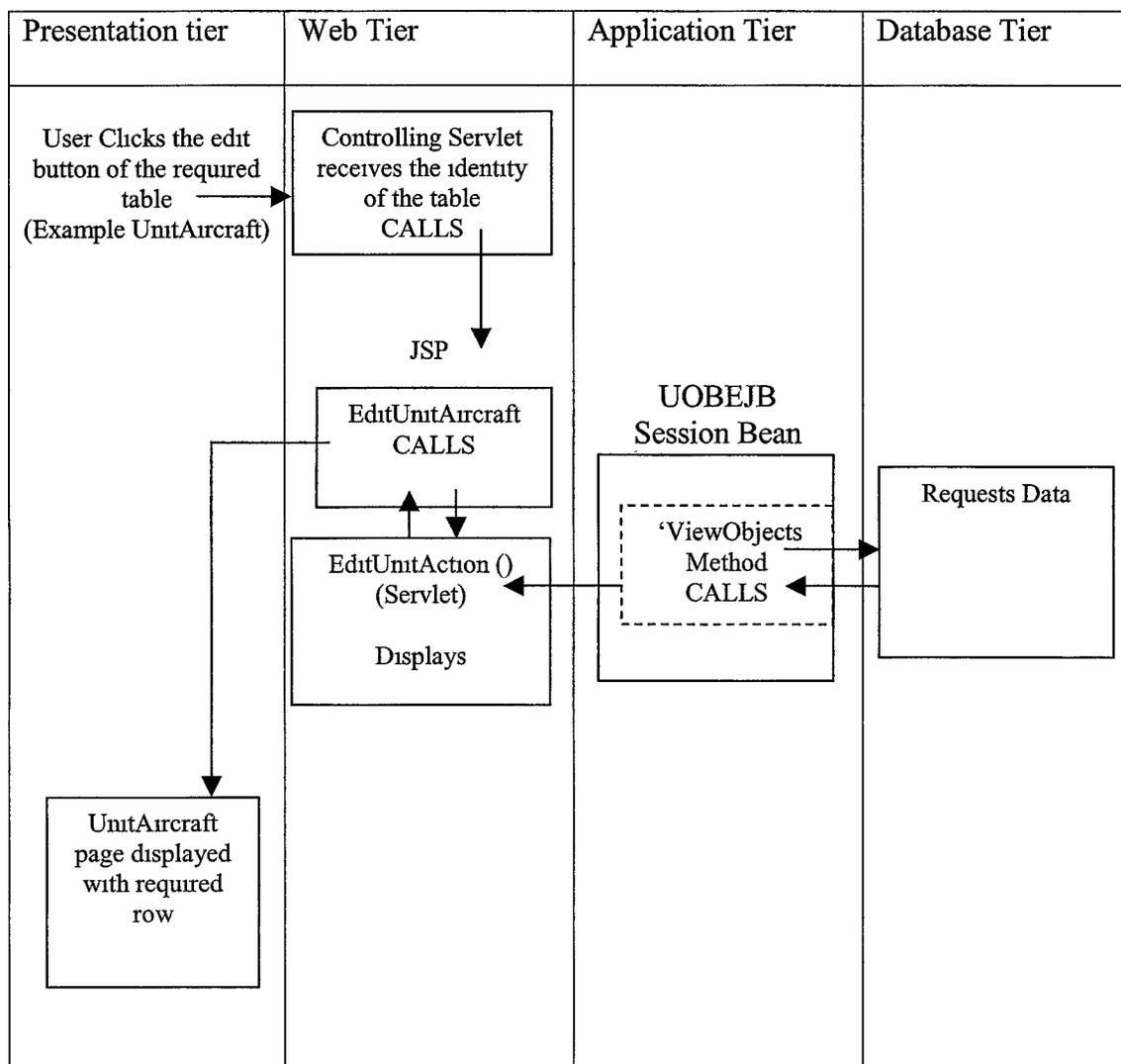
### 3. Editing data in a Table

#### ◆ Accessing the Table

##### A. High Level

The user selects a row in the Unit Aircraft table and clicks on the 'Edit' button of the Unit Aircraft table (figure 5.3).

##### B. Low Level



**Table 5.5: Accessing the Table to Edit Data**

- ◆ The Controlling Servlet (in the Web Tier) receives the identity of the table (Unit Aircraft) and the row number that the user has requested to edit.
- ◆ The Controlling Servlet invokes the JSP 'EditUnitAircraft.jsp'.
- ◆ EditUnitAircraft.jsp calls on the 'Edit Unit Aircraft Action Setup' servlet and requests the current contents of the row that has to be edited in the Unit Aircraft table.
- ◆ The Edit Unit Aircraft Action Servlet calls on the ViewObjects method of the UOBEJB (in the application tier), which in turn requests the data from the database. Upon receipt of data, the servlet sends the data to EditUnitAircraft.jsp.
- ◆ Using this data received, EditUnitAircraft.jsp displays the 'Edit Unit Aircraft' html page (Figure 5.6), with filled in, editable entries, each line corresponding to the columns of the row to be edited, in the Unit Aircraft table.

## Editing the Table

### A. High Level

The user edits the columns as desired and clicks the 'OK' button.

## Welcome to UOB Editor

### Edit Unit Aircraft

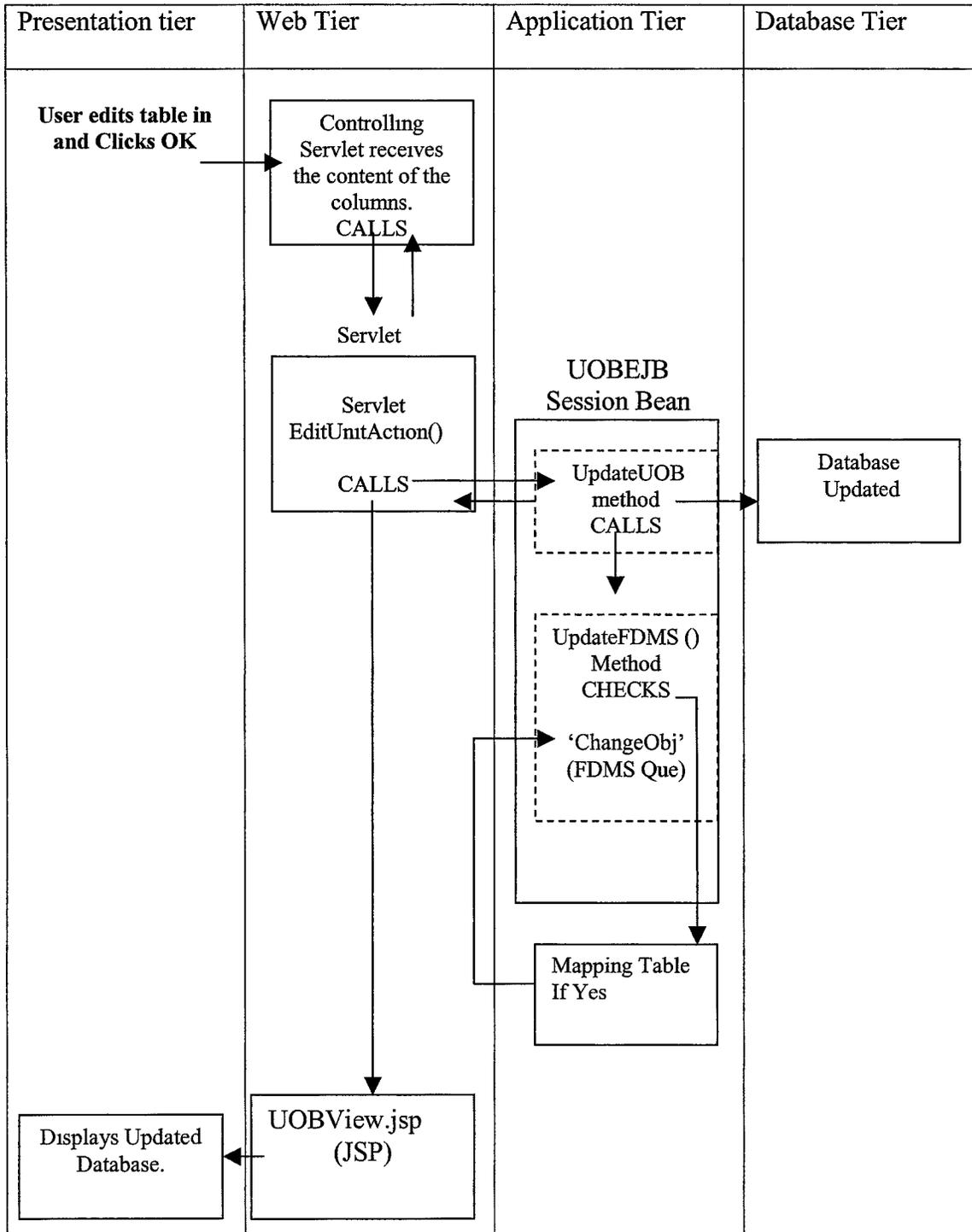
#### Unit Aircraft

UH1V	AIC
BN HQ	Unit Id
H13579	Aircraft Code
UH-1V Utility Helicopter	Aircraft Desc
2	Aircraft Qty Reqd
2	Aircraft Qty Auth

Ok

**Figure 5.6: Editing the Table**

**B. Low Level**



**Table 5.6: Editing the Data in the Table**

- ◆ The Controlling Servlet receives the edited contents of the row in the Unit Aircraft table.
- ◆ It calls the 'Edit Unit Aircraft Action' servlet and passes to it the edited row.
- ◆ The 'Edit Unit Aircraft Action' servlet calls the 'UpdateUOB' method of the UOBEJB bean in the Application Tier and passes to it the identity of the table (Unit Aircraft) and the edited contents of the row in this table.
- ◆ The UpdateUOB method sends the data to the database for updating the specified row in the Unit Aircraft table.
- ◆ The UpdateUOB method then calls the UpdateFDMS method in the UOBEJB to update the FDMS tables, if required.
- ◆ The UpdateFDMS method checks in a mapping table to see if the data is mapped into the FDMS database. If so, it creates a 'ChangeObj' object and inserts into it the identity of the operation ('update'), the identity of the table (Unit Aircraft), keys for identifying the row to be updated and the new data itself. Then, it pushes the 'ChangeObj' object to the queue of the FDMS system. When done, it returns control to the UpdateUOB method, which in turn, indicates to the Edit Unit Aircraft Action servlet that it's done.
- ◆ The 'Edit Unit Aircraft Action' servlet indicates to the Controlling Servlet that it is done
- ◆ The Controlling Servlet calls UobView.jsp. UobView.jsp gets all the current (including the edited row) contents of the database and displays them to the user in the UOB Editor html page. At this point, the system waits for new user input.

### 5.2.1.3 Logging Out

#### A. High Level

The user clicks on the 'Logout' button.

#### B. Low Level

- ◆ The Controlling Servlet (in the Web Tier) receives the message that the user has requested to log out.
- ◆ The Controlling Servlet invokes the JSP 'Login.jsp'.
- ◆ Login.jsp displays an html page (Figure 5.1) requesting a user to enter his User name and Password if he wishes to log back in.
- ◆ At this point, the system waits for new user input.

## 5.2.2 Component Modules of the Transmission Side (UOB)

### 5.2.2.1 The Presentation Tier

This consisted of the Microsoft Explorer browser on our system.

### 5.2.2.2 The Web Tier

The principle component modules of the web tier are Java Server Pages (JSP) and Servlets.

#### 5.2.2.2.1 JSPs in the Web Tier

##### a) Login.jsp

**Task:** This JSP displays a form to a user where he can enter a username and password

**Function:** This JSP contains html functionality to display two text boxes and a submit button. The first text box allows entry of a username. The second allows entry of a password. The submit button dispatches the response to the web tier.

**b) Errorlogin.jsp**

**Task:** Communicates login error to a user if username or password was invalid

**Function:** This is a very simple JSP. It contains text that displays a login error message to a user. It also contains a submit button with the text “OK”, allowing a user to communicate to the system that he has read the message.

**c) UobView.jsp**

**Task:** Displays contents of the UOB tables in tabular format

**Function:** This JSP creates four tables, one for each table in the UOB database – Unit, Unit Aircraft, Unit Equipment and Unit Personnel. For each table, the JSP calls on the viewObjects() method of the UobEJBBean in the application tier, communicating the name of the table it is currently working on, and requests all the data values in that table from the database, which it receives in an array. It then displays all the values from that array. For each entry in the table, the JSP displays a radio button, allowing selection of any entry in a table. For each table (except the UNIT table), it displays three buttons, Create, Delete and Edit, allowing a user to communicate what he intends to do next. The JSP also displays a logout button allowing a user to signal his desire to log out of the system.

**d) CreateUnitAircraft.jsp / CreateUnitPersonnel.jsp / CreateUnitEquipment.jsp**

**Task:** Allows a user to create a new record in a table (Unit Aircraft, Unit Equipment or Unit Personnel).

**Function:** The three JSPs are nearly identical and differ only in the format for their corresponding tables. Each JSP displays a set of empty text boxes, allowing a user to enter a set of values corresponding to a row in the appropriate (Aircraft, Personnel or

Equipment) table. Each also displays a submit button with the label “OK” allowing a user to submit the values when the input is complete.

**e) EditUnitAircraft.jsp / EditUnitEquipment.jsp / EditUnitPersonnel.jsp**

**Task:** Allows a user to edit one record in a table (Unit Aircraft, Unit Equipment or Unit Personnel)

**Function:** This set of (nearly identical) JSPs allows a user to edit one record of a table. It is invoked when a user selects a radio button in UobView.jsp, indicating the selection of a record in a table and clicks on the corresponding ‘Edit’ button. The JSP displays a set of text boxes, pre-populated with the contents of the record that the user elected to edit, allowing a user to edit it with new values. (The values are given to the JSP by the servlet EditUnitAircraftSetupAction, described later). It also displays a Submit button with the label “OK”, allowing the user to indicate when he is done with the edits.

**5.2.2.2.2 Servlets**

Servlets contain all the logic in the web tier. The system has several servlets, named according to the functionality they perform. Each servlet is associated with a form, appropriately named, that contains user-entered values for that functionality. For instance, the servlet LoginAction contains logic related to the logging in process by a user. This servlet has the form LoginForm associated with it, which stores the username and password values entered by a user for logging in. Each servlet is described below:

**a) LoginAction**

**Task:** Validate user login

**Function:** This servlet is called by the controlling servlet when a user submits a username and password while attempting to log in. It first checks if communication with

the application tier has been initialized. Being the first servlet called by the controlling servlet, it also serves the task of initializing communication with the application tier. Once the communication is setup, this servlet calls the `ValidateUser()` method of the `UobEJBBean` in the application tier, passing the user entered username and password values to it, for validation. It receives either “true” or “false” indicating success or failure and returns those values to the controlling servlet for further action.

#### **b) ViewAction**

**Task:** Process input from a user when a user clicks either a Create, Delete, Edit or Logout button on the View page.

**Function:** This servlet is called by the controlling servlet when a user clicks on either a Create, Delete, Edit or Logout button on the view page, generated by the JSP `UobView.jsp`. The view page displays all the values in the tables of the UOB database and offers a user the choice of creating a new entry in any of the tables (Create), modifying an existing entry in the database (Edit), deleting an existing entry in the database (Delete) or logging out (Logout). This servlet does not perform any logic of its own; it works as a dispatcher. If the user clicked on a Create button, it informs the Controlling Servlet the name of the table for which the Create button was clicked, so that the appropriate JSP (`CreateUnitAircraft.jsp`, `CreateUnitEquipment.jsp` or `CreateUnitPersonnel.jsp`, respectively) could be invoked to allow the user to enter columns for a new record of that table. If the user clicked a Delete button, it informs the Controlling Servlet the table name and the record number in that table, so it could call the servlet `DeleteAction`. If the user clicked on an Edit button, it informs the Controlling Servlet the name of the table and the record number, so it could call the appropriate

EditActionSetupServlet as defined below. If the user clicked the logout button, it informs the Controlling Servlet accordingly, allowing it to call Login.jsp, to terminate the session.

**c) DeleteAction**

**Task:** Delete a specific record from a table in the UOB database

**Function:** This servlet is called by the controlling servlet when the user clicks on the Delete button for a table. Unlike Create and Edit, all deletions for all tables are handled from this one servlet. When called, it first obtains from the ViewForm the number of the radio button clicked by the user. Based on the entries displayed, it makes a determination about the table name and the record number in that table that the user would like deleted. Once determined, it calls the method deleteUOB() in the session bean UobEJBBean of the application tier, passing to it as arguments the name of the table and the identity of the record, for deletion upon completion, it informs the Controlling Servlet accordingly.

**d) CreateUnitAircraftAction / CreateUnitEquipmentAction**

**/CreateUnitPersonnelAction**

**Task:** Create a new entry in the appropriate table.

**Function:** This servlet is called by the controlling servlet when a user enters data for the columns of a table, to create a new record in it, on the form generated by the JSPs CreateUnitAircraft.jsp, CreateUnitEquipment.jsp or CreateUnitPersonnel.jsp respectively. It obtains all the data entered by the user from the associated form and calls the method createUOB() in the session bean UobEJBBean of the application tier, passing to it as arguments the name of the table for which a new record is to be created and data for the new record. Upon completion, it informs the Controlling Servlet accordingly.

**e) EditUnitAircraftActionSetup / EditUnitEquipmentActionSetup****/EditUnitPersonnelActionSetup**

**Task:** User would like to edit data in a specific record in a table. This servlet reads that data from the database to supply to one of EditUnitAircraft.jsp, EditUnitEquipment.jsp or EditUnitPersonnel.jsp, to allow the user to interactively edit the data.

**Function:** This servlet is called by the Controlling Servlet when a user clicks on an Edit button in the view window, after selecting a specific record in a table to edit. This servlet obtains the data for that record from the database to allow the user to edit it. The servlet first obtains the identity of the record that was clicked for editing. Next, it calls the method viewObjects() in the UobEJBBean of the application tier, passing as an argument the name of the table that it would like the record for. When it receives the values for that record, it stores the values in its associated form and returns control to the Controlling Servlet. The Controlling Servlet calls one of EditUnitAircraft.jsp, EditUnitEquipment.jsp or EditUnitPersonnel.jsp, loading in the process the values in the associated form supplied by this servlet, for display to the user.

**f) EditUnitAircraftAction / EditUnitEquipmentAction / EditUnitPersonnelAction**

**Task:** Receive edited values in a record of a table and send them to the database so that the record in the table can be updated.

**Function:** This servlet is called by the Controlling Servlet when a user has edited values in one of EditUnitAircraft.jsp, EditUnitEquipment.jsp or EditUnitPersonnel.jsp. The edited values are stored in its associated form. When invoked, it reads all the updated values from the form and calls the method updateUOB() in the session bean

UobEJBBean of the application tier, passing as arguments the name of the table and the updated values. When done, it returns control to the Controlling Servlet.

### **g) Controlling Servlet**

**Task:** Main dispatcher and controller

**Function:** The Controlling Servlet is the central controlling and dispatching authority for our application in the web tier. When a user submits any information in the browser by clicking on a button, the web tier invokes this servlet. The servlet obtains all the user-entered information from the web tier and based on the state and the logic, calls the appropriate form to store the information in. It then calls an action servlet to take appropriate action (which obtains user-entered information from that form)). When the action is complete and control returns to this servlet, based on the state, it calls a JSP to display the next page to the user.

#### **5.2.2.3 The Application Tier**

All logic in the application tier resides in the session bean UobEJBBean.java. The methods of this bean receive requests from the servlets in the web tier and process them, communicating with the database in the process, either to obtain stored data or to update the data stored there. The methods are logically named, indicating their functionality. They are defined below.

**a) updateUOB ( Unit ) / updateUOB ( UnitAircraft ) / updateUOB ( UnitEquipment )/**

**updateUOB ( UnitPersonnel )**

**Task:** Update a record in a table in the UOB database with the values sent as an argument. Send a (synchronizing) update packet to the FDMS system if required.

**Function:** This is a set of overloaded methods, called by the EditAction servlets (one of EditUnitAircraftAction, EditUnitEquipmentAction or EditUnitPersonnelAction) in the web tier. After a user has edited values in a record in a table, the servlet calls this method, passing as an argument an object containing the updated values. The type of object determines which one of these methods is invoked. The method extracts those values from the object and calls the database multiple times, each time to update one column in the record. Each call to the database is done using a JDBC statement, with a string containing an embedded SQL command to update the appropriate record in the table with the specified column, thereby updating the record in the database for that table. Once the UOB database is updated, it calls the method updateFDMS() of the session bean to update the FDMS database if required, passing to it the name of the table, the operation (update) and the object containing the updated values.

**b) createUOB ( Unit ) / createUOB ( UnitAircraft )/ createUOB ( UnitEquipment )/  
createUOB ( UnitPersonnel )**

**Task:** Create a new record in a table in the UOB database with the values sent as an argument. Send a (synchronizing) update packet to the FDMS system if required.

**Function:** This is a set of overloaded methods, called by the CreateAction servlets (one of CreateUnitAircraftAction, CreateUnitEquipmentAction or CreateUnitPersonnelAction) in the web tier. After a user has entered values to create a new record in a table, the servlet calls this method, passing as an argument an object containing the column values for the new record. The type of object determines which one of these methods is invoked. The method extracts those values from the object and calls the database using a JDBC statement, with a string containing an embedded SQL

command to create a new record in the table using the values supplied, thereby creating a new record in the database for that table. Once the creation is done, it calls the method `updateFDMS()` of the session bean to update the FDMS database if required, passing to it the name of the table, the operation (create) and the object containing the new values.

**c) `deleteUOB ( Unit ) /deleteUOB ( UnitAircraft ) / deleteUOB ( UnitEquipment )`  
`deleteUOB ( UnitPersonnel )`**

**Task:** Delete a specific record in a table in the UOB database. Send a (synchronizing) update packet to the FDMS system if required.

**Function:** This is a set of overloaded methods, called by the `DeleteAction` servlet in the web tier. After a user has selected a record in a table and clicked on the Delete button, the servlet calls this method, passing as an argument an object containing the values for the record to delete. The type of object determines which one of these methods is invoked. The method extracts those values from the object and calls the database using a JDBC statement, with a string containing an embedded SQL command to delete the record in the specified table. Once the deletion is done, it calls the method `updateFDMS()` of the session bean to update the FDMS database if required, passing to it the name of the table, the operation (delete) and the object containing the old (deleted) values.

**d) `ReadUnitData() /ReadUnitAircraftData() /ReadUnitEquipmentData()`  
`ReadUnitPersonnelData()`**

**Task:** Read from the database all the records from the appropriate table.

**Function:** These are internal methods in the session bean, called from the method `viewObjects()`, defined below. Each method reads all the records from a table in the database, as per its name. The set of records is returned in an array of objects, each object

representing one record of the table. The method first sends a SQL query to the database, requesting the count (number) of records in the table. Upon receiving the count, it creates an array of that size. It then sends another SQL query to the database requesting all the records in the table. An object is created for each record in the table, containing all the columns of the record and stored in the array. When all the records have been read and stored in the array, it is returned to the viewObjects() method.

**e) viewObjects ( ClassName )**

**Task:** Read all the records of a table

**Function:** This method of the session bean reads all records of a table in the database.

The name of the table is specified by the ClassName argument. Depending on the name (Unit, Unit Aircraft, Unit Equipment or Unit Personnel), it calls one of ReadUnitData(), ReadUnitAircraftData(), ReadUnitEquipmentData or ReadUnitPersonnelData respectively. Upon receipt of data in an array of objects, it returns the data to the calling servlet in the web tier.

**f) checkMappingTable( TableName )**

**Task:** Check if records in this table are mapped in the FDMS database.

**Function:** This method is an internal method in the session bean, called by the updateFDMS() method of the bean, defined below. When updateFDMS() is called by any method of the bean, it has to make a determination whether to send a 'change' packet to the FDMS system. In turn, it calls the checkMappingTable() method, passing it the table name as an argument. This method checks the table name against an internal map and returns a 'true' or 'false' value, based on the mapping of the records in the table.

**g) updateFDMS ( TableName, Operation, Object)**

**Task:** Check if records in this table are mapped to FDMS. If so, create and send a 'change' object to the queue in the FDMS system.

**Function:** This is an internal method of the session bean UobEJBBean. It is called by the various update, create and delete methods of this bean to synchronize changes made in the UOB database with the FDMS database. This method first calls the checkMappingTable() method of the bean to check whether the operation on the table requires a synchronization with the FDMS database. If 'true', it creates a 'ChangeObj' object, storing within it the table name, the operation (update, create or delete) and a copy of the UOB object that was the subject of change. It then creates a session to connect with the FDMS system. Once it has a session object, it creates a publisher object to publish a message on the queue of the FDMS system. Once it has the publisher object, it publishes a message to the FDMS system, sending along with it the 'ChangeObj' object as an argument. Once done, it then closes the session.

**h) validateUser ( UserName, Password )**

**Task:** Validate a password for UserName

**Function:** This method is called by the LoginAction servlet in the web tier, to validate that the username and password entered by a user to access the system are valid. The method makes a SQL query (using JDBC) to get the password column, corresponding to the column Username in the Users table of the database. It then matches the password received from the web tier with the one obtained from the database. If they match, it returns 'true'. Else, it returns false.

**i) dbInit() / dbClose()**

**Task:** Initialize the close the connections with the database

**Function:** These methods initialize and close connections with the MySQL database.

Before any function makes a JDBC call embedding a SQL statement to the database, it calls on the dbInit() method to initialize the connection. When done, it closes the connection.

**5.2.2.4 The Database Tier**

The database tier we created for our purpose is consisted of the MySQL database system.

'Unit', 'Unit Aircraft', 'Unit Equipment' and 'Unit Personnel' were the four tables created on this system

The database was accessed from the application tier of our system (the various methods of our session bean, UobEEJBBBean) using JDBC calls. Each access to the database consisted of the following steps:

1. Establish a connection with the database.
2. Create a JDBC statement object.
3. Attach a string containing a SQL command to the statement object.
4. Execute the statement object.
5. Close the statement object.
6. Close the connection to the database.

The SQL commands are described under their sending methods.

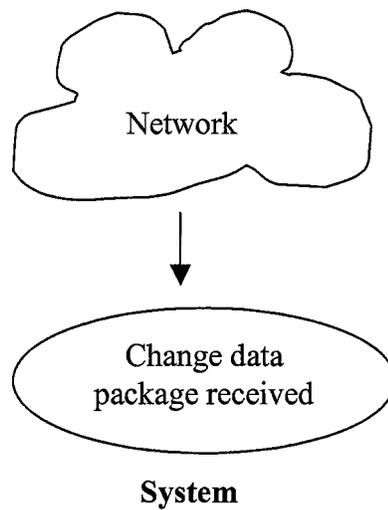
### 5.3 Receiving Side

#### 5.3.1 Implementation Description of Receiving Side Modules

##### 5.3.1.1 Data is accepted by the Receiving Side (FDMS)

###### A. High Level

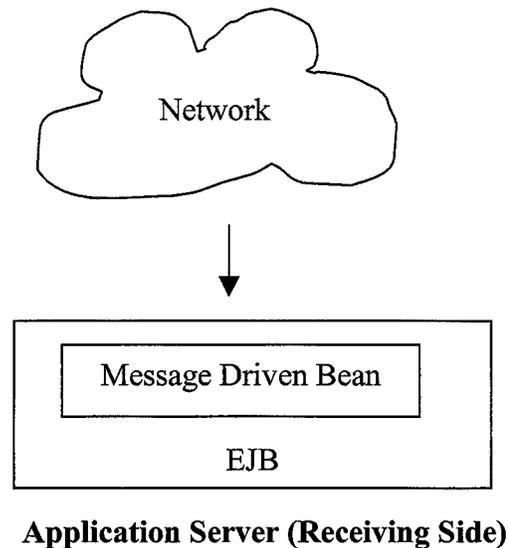
System Accepts the 'change' data sent from the Transmission Side (UOB), via a network.



**Figure 5.7: Data Accepted by Receiving Side (FDMS)**

## B. Low Level

A Message Driven Bean (kind of EJB), residing in the application server fundamentally does the receiving side work.



**Figure 5.8: Data Transferred to Application Server (FDMS)**

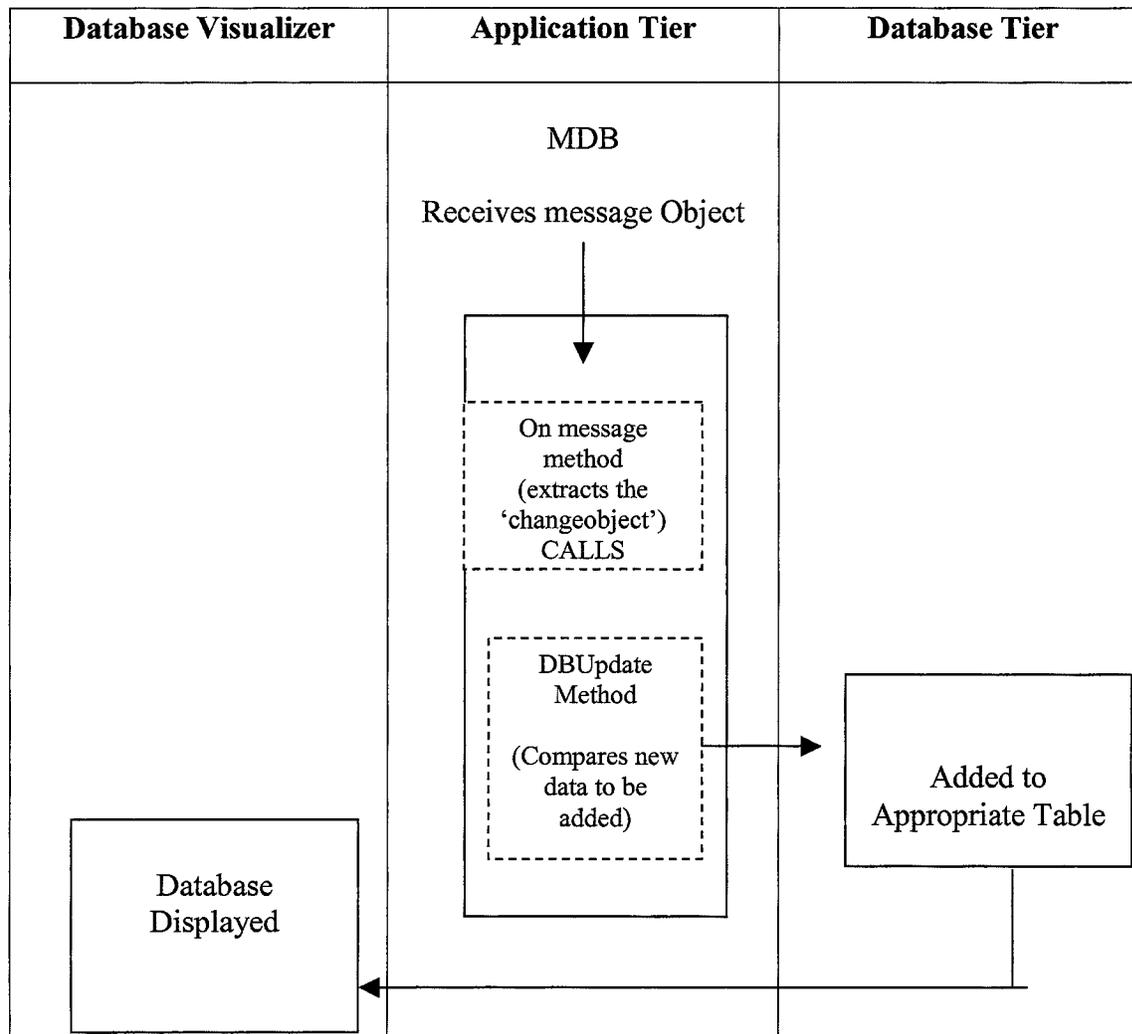
The MDB (Message Driven Bean) listens on the Message Queue for this system. In normal mode, it is passive and not in execution mode. When a message is received in this queue, the application tier invokes the MDB and calls its `OnMessage()` method, passing to it the contents of the message received.

### 5.3.1.2 Different Kinds of ‘Change’ Data Accepted

We will recall that when a user edits a UOB table (Create / Delete / Update), the `UpdateFDMS` method in the UOB EJB (in the application tier of that system) creates and loads a ‘ChangeObj’ packet and dispatches it to the Message Queue of the Receiving system.

Upon receipt of this packet, the application tier of this system wakes up the MDB (creates a session object of this bean), calls its OnMessage() method and passes to it the packet received in the queue.

### 1. The 'create' operation



**Table 5.7 Creating a New Row in Database**

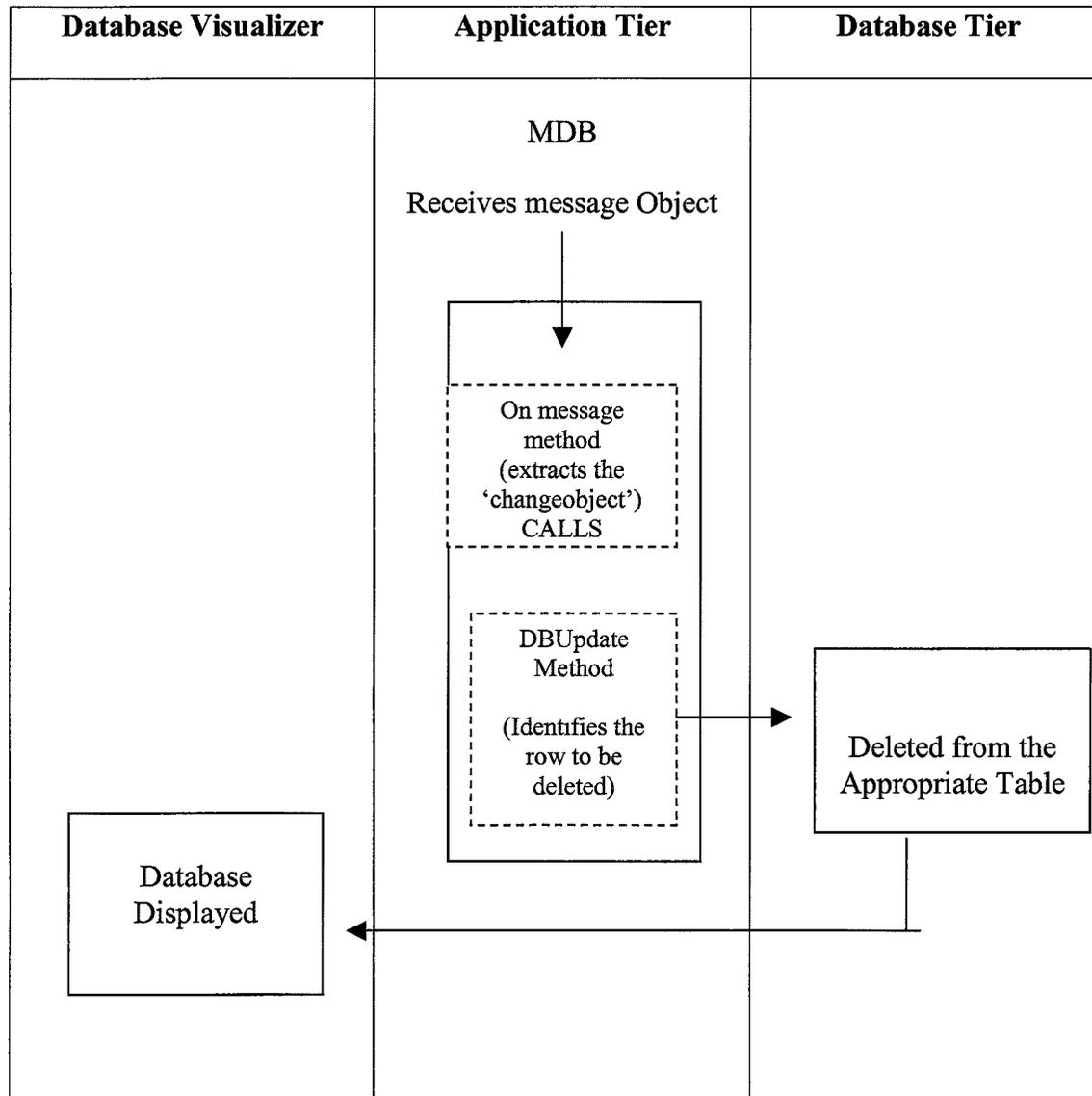
MDB receives a packet from the message queue. This packet contains a 'ChangeObj' object, specifying an operation to be performed. This operation is one of 'create', 'delete' or 'update'.

- ◆ For the 'create' operation, the MDB receives a message object containing a 'ChangeObj' object, which in turn contains the following data:
  1. Identity of the table in UOB where a row was created (example Unit Aircraft).
  2. Identity of the operation ('create').
  3. Data for insertion in the FDMS tables corresponding to the creation of a new row in the Unit\_Aircraft table in the UOB database. This is the data to be synchronized.
- ◆ The OnMessage method extracts the ChangeObj object from the message object. It then calls the DbUpdate method, passing the object to it.
- ◆ The DbUpdate method compares the new data to be added with the existing data in the tables to ensure that it's not going to duplicate an existing record. Specifically, it checks the AIC and the UIC of the incoming object with the 'Component Entity SDS ID' and 'Composite Entity SDS ID' of the Entity Component table in the FDMS database. If a record is found that matches this data, the entry is deemed to be duplicate and rejected. Else, it is accepted for addition within the FDMS tables.
- ◆ A new record is first created in the Entity Component Table. The 'Component-Entity-RDS-ID' value is deduced programmatically based on incrementing the highest existing value. The incoming AIC and the UIC represent the 'Component Entity SDS ID' and the 'Composite Entity SDS ID' respectively. The 'Composite-

Entity-RDS-ID' is obtained from the Component Entity table, based on the value of the 'Composite Entity SDS ID'. The Cardinality is maintained as '1'.

- ◆ Next, a new record is inserted into the Entity table. The 'Entity RDS ID' and the Entity-SDS-ID values correspond to the 'Component-Entity-RDS-ID' and the 'Component Entity SDS ID' values in the Entity Component table. The 'Entity Name' is derived from the 'Aircraft Description' in the incoming message object. The 'Entity Stereotype' and the 'Entity Type' are deduced from the identity of the UOB table created, in this instance the Unit Aircraft table.
- ◆ Finally, two new entries are inserted in the Entity Characteristic table. For each entry, the 'Entity RDS ID' and the Entity-SDS-ID correspond to the values in the Entity table. The Characteristic-RDS-ID and the Characteristic-SDS-ID are obtained from the FDMS Characteristic table, corresponding to the UOB table created – the Unit Aircraft table for Aircraft Quantity Required and Aircraft Quantity Authorized. The numeric data for each of these is contained within the incoming data in the ChangeObj object.
- ◆ This completes the data synchronization on the FDMS side for the 'create' operation.

## 2. The 'delete' operation

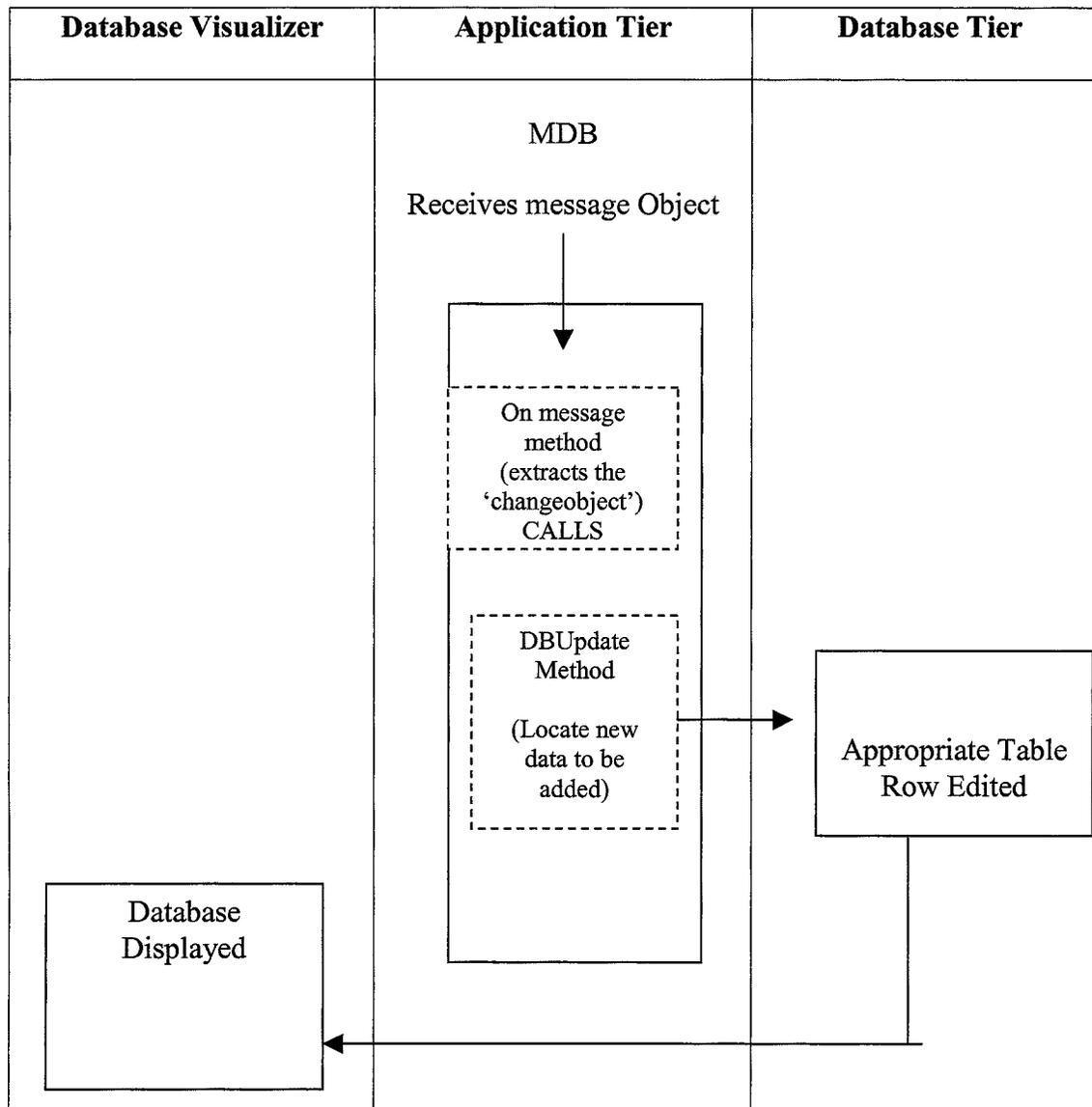


**Table 5.8: Deleting a Row in Database**

- ◆ For the **'delete'** operation, the MDB message bean receives a message object containing a **'ChangeObj'** object, which in turn contains the following data:
  1. Identity of the table in the UOB where a row was deleted (assume Unit Aircraft).

2. Identity of the operation ('delete').
  3. Data corresponding to the deleted row in the Unit\_Aircraft table, in the UOB database.
- ◆ The OnMessage method extracts the ChangeObj object from the message object. It then calls the DbUpdate method, passing the object to it.
  - ◆ The DbUpdate method identifies and deletes corresponding rows in the Entity Component, the Entity and the Entity Characteristic tables of the FDMS database as follows.
  - ◆ The first table selected is the Entity Component table. The incoming AIC and the UIC correspond to the 'Component Entity SDS ID' and the 'Composite Entity SDS ID' respectively, in this table. Together, they form a unique identity for each row in the table. The row corresponding to the incoming values is found, giving us the Component-Entity-RDS-ID. This row of the table is deleted.
  - ◆ The next table selected is the Entity table. The 'Component-Entity-RDS-ID' obtained in the previous step from the Entity Component table corresponds to the 'Entity RDS ID' of this table and identifies each row uniquely. The row is found and deleted.
  - ◆ The last table is the Entity Characteristic table. The 'Component-Entity-RDS-ID' of the Entity Component table corresponds to the 'Entity RDS ID' of this table. The table contains two rows corresponding to this value. Both are deleted.
  - ◆ This completes the data synchronization on the FDMS side for the 'delete' operation.

### 3. The 'update' (Edit-Data) operation



**Table 5.9: Updating (Edit-Data) the Database**

- ◆ For the **'update'** operation, the MDB receives a message object containing a ChangeObj object, which in turn contains the following data:

1. Identity of the table in the UOB where a row was updated (assume Unit Aircraft).
  2. Identity of the operation ('update').
  3. Data corresponding to the updated row in the Unit\_Aircraft table, in the UOB database.
- ◆ The OnMessage method extracts the ChangeObj object from the message object. It then calls the DbUpdate method, passing the object to it.
  - ◆ The DbUpdate method identifies and updates corresponding rows in the Entity Characteristic table of the FDMS database as follows.
  - ◆ The identity of the rows to be updated in the Entity Characteristic table is partly determined from the Entity Component table. The incoming AIC and the UIC correspond to the 'Component Entity SDS ID' and the 'Composite Entity SDS ID' respectively, in the Entity Component table. Together, they form a unique identity for each row in the table. The row corresponding to the incoming values is found, giving us the Component-Entity-RDS-ID.
  - ◆ The identity of the table updated (Unit Aircraft) gives us the Characteristic-RDS-IDs from the Characteristic table, corresponding to Aircraft Quantity Required and the Aircraft Quantity Authorized. Two values for Characteristic-RDS-IDs are obtained.
  - ◆ The 'Component-Entity-RDS-ID' of the Entity Component table corresponds to the 'Entity RDS ID' of the Entity Characteristic table. This value and the Characteristic-RDS-IDs obtained from the Characteristic table form unique keys for identifying the rows to be updated in the Entity Characteristic table. The Entity-Characteristic-

Numeric-Value for each of these rows is updated with the incoming values for Aircraft Quantity Required and Aircraft Quantity Authorized.

- ◆ This completes the data synchronization on the FDMS side for the ‘update’ operation.

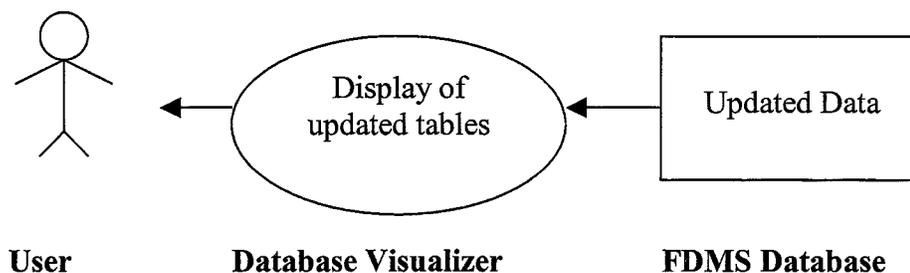
### **FDMS Data displayed to the User.**

The contents of the FDMS database are displayed to the user using an application, the Database Visualizer. This application is in the public domain and was downloaded specifically for this purpose.

#### ◆ **Setting Up the Database Visualizer**

The setup of the application is extremely simple. It is interactively set up to identify the database drivers to use for the MySQL database on our system. Additionally, the name of the database and the tables to be viewed are specified. The application automatically connects to the database and displays the tables.

For our purpose, we set it up with the Entity, Entity Component, Characteristic and the Entity Characteristic tables of the FDMS database. All changes made to these tables could be viewed in this manner.



**Figure 5.9: FDMS Data Display**

## **5.3.2 Component Modules of the Receiving Side (FDMS)**

### **5.3.2.1 Database Visualizer**

It is interactively set up to identify the database drivers to use for the MySQL database on our system. Additionally, the name of the database and the tables to be viewed are specified. The application automatically connects to the database and displays the tables.

### **5.3.2.2 Application Tier**

All logic in the application tier of the FDMS system resides in one **Message Driven Bean**, DbUpdate. This bean subscribes as a 'Listener' to the queue in the application server of the FDMS system such that, when a message is received in the queue, its onMessage() method is called by the application server. The onMessage() method, described below, calls other methods of DbUpdate, to update the FDMS database, based on the contents of the message received.

#### **a) OnMessage()**

**Task:** Decode the incoming 'change' message packet to extract the 'ChangeObj' object sent by the FDMS system

**Function:** This method is invoked by the application server upon receipt of a message in the queue. The message packet is passed as an argument to the method. Upon receipt, the method first isolates and extracts the ChangeObj object sent by the UOB system. Once done, the method checks the operation involved in the synchronization and based on it, calls one of dbUpdate(), dbDelete() or dbCreate() methods in the message driven bean.

#### **b) DbUpdate(ChangeObj )**

**Task:** Update the FDMS database corresponding to the modification (editing) of an existing record in a UOB table, mapped to entries in the FDMS database

**Function:** This function is invoked by the onMessage() method of the MDB. The change object, ChangeObj is passed to it as an argument. We will describe the program process for a change in the Unit Aircraft table of the UOB database. The process for the Unit Equipment and the Unit Personnel tables is identical.

The mapped values between the UOB and FDMS databases happen to be ‘Quantity Required’ and ‘Quantity Authorized’. Since the entire UOB record is sent in the ‘ChangeObj’ object, these columns are first obtained from the incoming object. These are the values that will ultimately be updated in the FDMS tables.

- ◆ The method first obtains the ‘Component-Entity-RDS-ID’ from the ‘Entity Component’ table of FDMS, using the incoming AIC column as the “‘Component Entity SDS ID’” column and the incoming UIC column as the “‘Composite Entity SDS ID’” column of the ‘Entity Component’ table, as primary keys. The database is accessed using SQL queries embedded in JDBC statements, identical to the way it was used in the UOB system.
- ◆ It then obtains the ‘Characteristic-RDS-IDs’ from the Characteristic table of FDMS, corresponding to ‘Characteristic-Names’ ‘Aircraft Quantity Required’ and ‘Aircraft Quantity Authorized’.
- ◆ Finally, it updates two ‘Entity-Characteristic-Numeric-Value’ columns in the Entity Characteristic table using the ‘Entity Component RDS ID’ as ‘Entity RDS ID’ and each ‘Characteristic Component RDS ID’ as ‘Characteristic-RDS-ID’ used as identifying keys to uniquely identify the records to update.
- ◆ The program process corresponding to the editing of Unit Equipment and Unit Personnel tables is nearly identical to the one described above. The only difference is

in the Characteristic-RDS-IDs obtained from the Characteristic table; they are obtained for Equipment/Personnel Quantity Required and Equipment/Personnel Quantity Authorized. All else is identical.

- ◆ This completes the synchronization of the FDMS system corresponding to the editing of UOB tables.

#### **d) DbDelete (ChangeObj )**

**Task:** Update the FDMS database corresponding to the deletion of an existing record in a UOB table, mapped to entries in the FDMS database.

**Function:** This function is invoked by the onMessage() method of the MDB. The change object, ChangeObj is passed to it as an argument. We will describe the program process for a record deletion in the Unit Aircraft table of the UOB database. The process for the Unit Equipment and the Unit Personnel tables is identical.

The mapped values between the UOB and FDMS databases happen to be ‘Quantity Required’ and ‘Quantity Authorized’. Since the entire UOB record is sent in the ‘ChangeObj’ object, these columns are first obtained from the incoming object. The records corresponding to these values will be deleted in the Entity Characteristic FDMS table. In addition, entries will be deleted from the Entity and Entity Component tables as below.

- ◆ The method first obtains the ‘Component-Entity-RDS-ID’ from the ‘Entity Component’ table of FDMS, using the incoming AIC column as the ‘‘Component Entity SDS ID’’ column and the incoming UIC column as the ‘‘Composite Entity SDS ID’’ column of the ‘Entity Component’ table, as primary keys. After obtaining the ‘Component-Entity-RDS-ID’, the record corresponding to this column in the

Entity Component table is deleted. The database is accessed using SQL queries embedded in JDBC statements, identical to the way it was used in the UOB system.

- ◆ It then deletes a record in the Entity table, using the ‘Component-Entity-RDS-ID’ obtained earlier as the ‘Entity RDS ID’ in this table, as the identifying key.
- ◆ It then obtains the ‘Characteristic-RDS-IDs’ from the Characteristic table of FDMS, corresponding to ‘Characteristic-Names’ ‘Aircraft Quantity Required’ and ‘Aircraft Quantity Authorized’.
- ◆ Finally, it deletes two records in the ‘Entity Characteristic’ table using the ‘Entity Component RDS ID’ as ‘Entity RDS ID’ and each ‘Characteristic Component RDS ID’ as ‘Characteristic-RDS-ID’, used as identifying keys to identify the records to delete.
- ◆ The program process corresponding to the deletion of a record in the Unit Equipment and Unit Personnel tables is nearly identical to the one described above. The only difference is in the Characteristic-RDS-IDs obtained from the Characteristic table; they are obtained for Equipment/Personnel Quantity Required and Equipment/Personnel Quantity Authorized. All else is identical.
- ◆ This completes the synchronization of the FDMS system corresponding to the deletion of a record in one of the UOB tables.

**e) DbCreate( ChangeObj )**

**Task:** Update the FDMS database corresponding to the creation of a new record in a UOB table, mapped to entries in the FDMS database.

**Function:** This function is invoked by the onMessage() method of the MDB. The change object, ChangeObj is passed to it as an argument. We will describe the program process

for a new record creation in the Unit Aircraft table of the UOB database. The process for the Unit Equipment and the Unit Personnel tables is identical.

As before, incoming values for 'Aircraft Quantity Required' and 'Aircraft Quantity Authorized' are first obtained from the incoming object.

- ◆ The method first checks in the 'Entity Component' table to make sure that an incoming record supposed to be new does not already exist in the table. Using the incoming 'AIC' column as the 'Component Entity SDS ID' and the incoming 'UIC' column as the "Composite Entity SDS ID", used as unique identifying columns, it tries to find an existing record with these values. If a record already exists, the incoming creation is rejected and the method returns. If not, the method continues.
- ◆ The method next creates a new record in the 'Entity Component' table. To do so, it must first have a unique value for the 'Component-Entity-RDS-ID', not already in the table. The method reads all values in this table and finds the highest existing value corresponding to Aircraft type records. Once the highest value is found in the table, it increments it by one to get the new value for the 'Component-Entity-RDS-ID'.
- ◆ The method next obtains the 'Component-Entity-RDS-ID' from the Entity Component table, using the incoming UIC as the "Component Entity SDS ID". This becomes the 'Composite-Entity-RDS-ID' for the new record to be created.
- ◆ Finally, cardinality is selected as 1 for the operation involved.
- ◆ The method now has all the columns needed to create a new record in the Entity Component table. It makes a SQL call to the database to do so. This updates the Entity Component table.

- ◆ The method next creates a new record in the Entity table of FDMS. It already has two columns for the new record – the ‘Component-Entity-RDS-ID’ and the ‘‘Component Entity SDS ID’’ of the ‘Entity Component’ table are used as the ‘Entity RDS ID’ and the ‘Entity SDS ID’ for this table, respectively. The ‘Entity Name’ column is obtained from the incoming ‘Aircraft Description’ column of the ChangeObj object. Given the nature of the record (Aircraft), the ‘Entity Stereotype’ column is chosen as ‘‘Equipment’’ and the ‘Entity Type’ column is chosen as ‘‘Aircraft’’. With this, the method now has all the columns needed for the creation of a new record in the Entity table. It makes a SQL call to do so. This updates the Entity table.
- ◆ The method finally creates two new entries in the Entity Characteristic table. To do so, it first obtains the ‘Characteristic-RDS-IDs’ corresponding to ‘Characteristic-Names’ ‘Aircraft Quantity Required’ and ‘Aircraft Quantity Authorized’ in the Characteristic table. These will be the ‘Characteristic-RDS-IDs’ for the two new entries in the ‘Entity Characteristic’ table. The ‘Characteristic-SDS-ID’ and the ‘Entity-Characteristic-String-Value’ for each record is null. The ‘Entity RDS ID’ and the ‘Entity SDS ID’ for each record correspond to the ‘Entity RDS ID’ and the ‘Entity SDS ID’ used for the new record in the Entity table. With this, the method has all the columns needed for the creation of two new records. The first record is created with the values specified and the ‘Entity-Characteristic-Numeric-Value’ corresponding to the incoming ‘Aircraft Quantity Required’ value. The second is created with the same values and with ‘Entity-Characteristic-Numeric-Value’ equal

to the incoming 'Aircraft Quantity Authorized' value. The method makes a SQL call to create the records in the Entity Characteristic table.

The program process corresponding to the creation of a new record in the Unit Equipment and Unit Personnel tables is nearly identical to the one described above. The only difference is in the Characteristic-RDS-IDs obtained from the Characteristic table; they are obtained for Equipment/Personnel Quantity Required and Equipment/Personnel Quantity Authorized. All else is identical.

This completes the synchronization of the FDMS system corresponding to the creation of a new record in one of the UOB tables.

### **5.3.2.3 Database Tier**

The database tier we created for our purpose consisted of the MySQL database system. 'Entity', 'Entity-Component', 'Characteristic' and 'Entity-Characteristic' were the four tables created on this system

The database was accessed from the application tier of our system using JDBC calls.

Each access to the database consisted of the following steps:

1. Establish a connection with the database.
2. Create a JDBC statement object.
3. Attach a string containing a SQL command to the statement object.
4. Execute the statement object.
5. Close the statement object.
6. Close the connection to the database.

The SQL commands are described under their sending methods

## **5.4 Summary**

This chapter describes the high-level and the low-level implementation process of database synchronization that the user and the system go through.

The next chapter draws conclusions for the work we have performed, and suggests possible future work.

## CHAPTER 6

### CONCLUSIONS AND FUTURE WORK

#### 6.1 Conclusions

All needs and requirements of our project were met through the execution of our systems.

The databases synchronized successfully for all operations performed.

The J2EE framework provided an excellent system for the execution of this project. It provided all the tools needed for the various aspects our two systems:

- ◆ Provided a Web Tier for the presentation of html pages to the user, thereby providing an easy to use, interactive platform for user interaction.
- ◆ Provided a sophisticated Application Tier that allowed us to contain all logic in one location. It communicated with the database for all our needs. Additionally, it provided us with a sophisticated messaging mechanism that was so vital for the success of our project. This mechanism allowed us to synchronize the data in separate databases through the Internet.
- ◆ It provided us with a fully developed database to take care of all our data needs.
- ◆ Above all, all the systems we used were available free, through a general public license.

## 6.2 Future Work

### ◆ **Perform on the Actual UOB and FDMS Databases**

Although the project was successful, it was simulated due to the unavailability of the actual UOB and FDMS databases to us. The system should be used on the real databases and enhanced to provide full synchronization on these databases.

### ◆ **Security Issues**

No identity validation was performed for this synchronization. Hence, any client that knows the Internet Protocol address of the messaging queue could send a message packet to alter the contents of the FDMS database, thereby corrupting the data. The UOB side system should perform an identity validation.

### ◆ **Confirmation can be sent back to the transmitting side about the receiving of the data.**

There is currently, no acknowledgement from the FDMS system to the UOB system about messages received. Hence, if the message queue was unavailable or overflowed, data could be lost. The system should be enhanced to account for transmission errors.

## GLOSSARY

**API** - Application Programming Interface

When applied to Java programming language, this is a set of classes and interfaces that specify a particular functionality.

**Asynchronous Messaging** - allows applications or components to communicate with other applications or components by exchanging messages in such a way that senders are independent of receivers. The sender sends its message and does not need to wait for the receiver to receive or process the message. The sender's message is delivered when the server is rebooted.

**Business Logic** – The code that implements the functionality of an application. In the Enterprise JavaBeans model, this logic is implemented by the methods of an enterprise bean.

**Business Tier** – The set of machines on which the business components execute in an n-tier application.

**Component** - A component in J2EE architecture is a grouping of functionality that forms a coherent unit. This unit can be deployed in a component container independently of other components. Applications can then be built by calling on the functionality of multiple, specialist components.

**Containers** – A container provides services for a component. These services can include lifecycle management, security, connectivity, transactions, and persistence. Each type of J2EE component is deployed into its own type of J2EE container.

**Database** – provides the basic storage and access to the organizations data.

**Deployment** - The process whereby software is installed into an operational environment.

**EJB** - Enterprise JavaBeans.

EJB is a component technology that helps developers create business object in the middle tier. These business objects consist of columns and methods that implement business logic. EJBs are building blocks of enterprise systems. They perform specific tasks by themselves or forward operations to other enterprise beans. EJBs are under control of the J2EE application server.

**EJB Container** – Manages the execution of enterprise beans for J2EE applications. A J2EE server provides an EJB container.

**FDMS** - Functional Descriptions of the Mission Space Model.

Functional mission space models are simulation implementation-independent functional descriptions of the real-world processes, entities, environmental factors, and their associated relationships and interactions within the context of a set of military missions, operations, or tasks.

**FDMS DIF** – FDMS Data Interchange Format.

**HTML** – HTML is the language used to define Web Pages that display in a Web browser.

**HTTP** – Hypertext Transfer Protocol. The Internet protocol used to fetch hypertext objects from remote hosts. HTTP messages consist of (architecture independent display specifications) requests from client to server and responses from server to client.

**IP** – Internet Protocol

**J2EE** – Java 2 Enterprise Edition.

J2EE is an environment for developing and deploying enterprise applications. The J2EE platform consists of a set of services, application programming interfaces, and protocols that provide the functionality for developing multitiered, Web-based applications.

**J2EE Server** – The runtime portion of a J2EE product. A J2EE server provides Web and EJB containers.

**JBOSS** – Integrates data in the Application Server.

**JDBC** – Java Database Connectivity.

JDBC technology allows an application component provider to perform connection and authentication to a database server, manage transactions, move SQL statements to a database engine for preprocessing and execution and execute stored procedures.

**JMS** – Java Messaging Service.

JMS provides a single standard, unified message API for five different message formats (including XML). It encapsulates and exchanges asynchronous messages reliably between enterprise Java applications regardless of the operating systems, platform, architecture, and computer languages being used.

**JMS API** – allows applications and components to create, send, receive, and read messages. It enables communication that is loosely coupled, asynchronous, and reliable

and allows applications and components written in the Java programming language to communicate with applications that use other messaging implementations.

**JMS Provider** – JMS provider is a server that provides messaging services such as routing messages, persistent message handling, and destination service to both the producer and receiver of a given message. It implements the JMS API for an enterprise messaging system and provides access to the services provided by the underlying message system.

**JNDI** – Java Naming Directory Interface.

JNDI is a directory that provides naming and directory functionality. It provides applications with methods for performing standard directory operations, such as associating attributes with objects and searching for objects using their attributes. Using JNDI, an application can store or retrieve any type of named Java object.

**JSP** – Java Server Pages.

An extensible Web technology that uses template data, custom elements, scripting languages, and server-side Java objects to return dynamic content to a client.

**Multitier** – A business system structured with two or more tiers or layers. The distributed application is generally defined with 3 layers, the presentation, business (application) and database. Applications can have many tiers each of which can be some specialization and three tiers listed.

**Protocol** – Protocol is precise set of rules defining how computers communicate: the format of addresses, how data is split into packets, etc. In networking terms, a protocol is used to address and ensure delivery of packets across a network.

**RMI** - Remote Method Invocation

RMI allows a Java object that executes on one machine to invoke a method of a Java object that executes another machine. It also provides support for network calls used by EJB in the J2EE architecture. RMI is a simple and powerful way to write distributed applications, but it works only in Java environment.

**Servlets** – A servlet is a component that extends the functionality of a Web server in a portable and efficient manner. It is written entirely in Java language. Extracts logic information from incoming html, and inserts variable information into outgoing html.

A servlet is a program that executes the code.

**SQL** – Structured Query Language. Also known as SEQUEL (Structured English Query Language). SQL is a language used to communicate messages to the database server to create, update, retrieve, delete and manage data in the database.

**TCP** – Transmission Control Protocol.

**SQL database** – is a collection of tables and indexes to store various kinds of data in.

**Tiers** - A tier is a logical partition of the separation of concerns in the system. They are logical machines that may or may not be on separate physical machines.

Tiered structuring allows wrapping of functionality in familiar application programming interfaces, and independent development and maintenance of separate tiers.

**Tomcat** - The Tomcat servlet engine is an open-source Java implementation delivered by the Apache software Foundation. Tomcat can run as a standard server, or it can be plugged into most Web servers.

**UOB** – Unit Order of Battle.

UOB employs authoritative unit order of battle data to select the forces and to task-organize the functional mission requirements.

**Web Component** – A component is a software entity that provides a response to a request, either a servlet or a JSP page.

**Web Container** Manages the execution of JSP page and servlet components for J2EE applications. Web components and their container run on the J2EE server.

## APPENDIX A

### UOB DATABASE TABLES

**Table-1: Unit**

UNIT-IDENTIFICATION-CODE	PARENT-UNIT-IDENTIFICATION-CODE	UNIT-NAME	HOME-NAME	SHIP-CATEGORY	COUNTRY-CODE
ARMR BN		Armor Battalion	Blue	N1	US
BN HQ	ARMR BN	Armor Battalion Headquarters	Red	N2	US
ARMR CO	ARMR BN	Armor Company	Brown	N3	US
CO HQ	ARMR CO	Armor Company Headquarters	White	N4	US
ARMR PLT	ARMR CO	Armor Platoon	Orange	N5	US
CMD SN	ARMR PLT	Armor Command Section	Green	N6	US
ARMR SN	ARMR PLT	Armor Section	Yellow	N7	US

**Table-2: Unit-Personnel**

PIC	UNIT-IDENTIFICATION-CODE	PERSONNEL-DESCRIPTION	PERSONNEL-QUANTITY-REQUIRED	PERSONNEL-QUANTITY-AUTHORIZED
LT COL	BN HQ	Lieutenant Colonel	1	1
MAJ	BN HQ	Major	1	1
DRIVER	BN HQ	Tank Driver	2	2
GUNNER	BN HQ	Gunner	2	2
LOADER	BN HQ	Loader	2	2
PILOT	BN HQ	UH-1 Pilot	2	2
CAPT	CO HQ	Captain	1	1
1LT	CO HQ	First Lieutenant	1	1
DRIVER	CO HQ	Tank Driver	2	2
GUNNER	CO HQ	Gunner	2	2
LOADER	CO HQ	Loader	2	2
PILOT	COHQ	UH-1 Pilot	2	2
2LT	CMD SN	Second Lieutenant	1	1
DRIVER	CMD SN	Tank Driver	1	1
GUNNER	CMD SN	Gunner	1	1
LOADER	CMD SN	Loader	1	1
CHIEF	ARMR SN	Crew Chief	1	1
DRIVER	ARMR SN	Tank Driver	1	1
GUNNER	ARMR SN	Gunner	1	1
LOADER	ARMR SN	Loader	1	1

**Table-3: Unit-Equipment**

EIC	UNIT-IDENTIFICATION-CODE	EQUIPMENT-CODE	EQUIPMENT-DESCRIPTION	EQUIPMENT-QUANTITY-REQUIRED	EQUIPMENT-QUANTITY-AUTHORIZED
M1A1	BN HQ	M1234	Main Battle Tank	2	2
M23	BN HQ	M2345	7 62 MM Sidearm	2	2
M1R	BN HQ	M1357	Rifle	6	6
M1A1	CO HQ	M1234	Main Battle Tank	2	2
M23	CO HQ	M2345	7.62 MM Sidearm	2	2
M1R	CO HQ	M1357	Rifle	6	6
M1A1	CMD SN	M1234	Main Battle Tank	1	1
M23	CMD SN	M2345	7.62 MM Sidearm	1	1
M1R	CMD SN	M1357	Rifle	3	3
M1A1	ARMR SN	M1234	Main Battle Tank	1	1
M23	ARMR SN	M2345	7.62 MM Sidearm	1	1
M1R	ARMR SN	M1357	Rifle	3	3

**Table-4: Unit-Aircraft**

<b>AIC</b>	<b>UNIT- IDENTIFICATION- CODE</b>	<b>AIRCRAFT- CODE</b>	<b>AIRCRAFT- DESCRIPTION</b>	<b>AIRCRAFT- QUANTITY- REQUIRED</b>	<b>AIRCRAFT- QUANTITY- AUTHORIZED</b>
UH1V	BN HQ	H13579	UH-1V Utility Helicopter	2	2
UH1V	CO HQ	H13589	UH-1V Utility Helicopter	4	4

## APPENDIX B

### FDMS DATABASE TABLES

**TABLE-1: ENTITY**

Entity RDS ID	Entity SDS ID	Entity Name	Entity Stereotype	Entity Type
102	BN HQ	Armor Battalion Headquarters	Organization	Unit
103	ARMR CO	Armor Company	Organization	Unit
104	CO HQ	Armor Company Headquarters	Organization	Unit
105	ARMR PLT	Armor Platoon	Organization	Unit
106	CMD SN	Armor Command Section	Organization	Unit
107	ARMR SN	Armor Section	Organization	Unit
401	LT COL	Lieutenant Colonel	Person	Personnel
402	MAJ	Major	Person	Personnel
403	DRIVER	Tank Driver	Person	Personnel
404	GUNNER	Gunner	Person	Personnel
405	LOADER	Loader	Person	Personnel
406	PILOT	UH-1 Pilot	Person	Personnel
407	CAPT	Captain	Person	Personnel
408	1LT	First Lieutenant	Person	Personnel
409	DRIVER	Tank Driver	Person	Personnel
410	GUNNER	Gunner	Person	Personnel
411	LOADER	Loader	Person	Personnel
412	PILOT	UH-1 Pilot	Person	Personnel
413	2LT	Second Lieutenant	Person	Personnel
414	DRIVER	Tank Driver	Person	Personnel
415	GUNNER	Gunner	Person	Personnel
416	LOADER	Loader	Person	Personnel
417	CHIEF	Crew Chief	Person	Personnel
418	DRIVER	Tank Driver	Person	Personnel
419	GUNNER	Gunner	Person	Personnel
420	LOADER	Loader	Person	Personnel
801	M1A1	Main Battle Tank	Equipment	Equipment

(Table 1 Continued Next Page)

Table 1, continued.

802	M23	7 62 MM Sidearm	Equipment	Equipment
803	M1R	Rifle	Equipment	Equipment
804	M1A1	Main Battle Tank	Equipment	Equipment
805	M23	7.62 MM Sidearm	Equipment	Equipment
806	M1R	Rifle	Equipment	Equipment
807	M1A1	Main Battle Tank	Equipment	Equipment
808	M23	7 62 MM Sidearm	Equipment	Equipment
809	M1R	Rifle	Equipment	Equipment
810	M1A1	Main Battle Tank	Equipment	Equipment
811	M23	7.62 MM Sidearm	Equipment	Equipment
812	M1R	Rifle	Equipment	Equipment
1201	UH1V	UH-1V Utility Helicopter	Equipment	Aircraft
1202	UH1V	UH-1V Utility Helicopter	Equipment	Aircraft

Table-2: Entity-Component

Component Entity RDS ID	Component Entity SDS ID	Composite Entity RDS ID	Composite Entity SDS ID	Entity Component Cardinality
102	BN HQ	101	ARMR BN	1
103	ARMR CO	101	ARMR BN	1
104	CO HQ	103	ARMR CO	1
105	ARMR PLT	103	ARMR CO	1
106	CMD SN	105	ARMR PLT	1
107	ARMR SN	105	ARMR PLT	1
401	LT COL	102	BN HQ	1
402	MAJ	102	BN HQ	1
403	DRIVER	102	BN HQ	1
404	GUNNER	102	BN HQ	1
405	LOADER	102	BN HQ	1
406	PILOT	102	BN HQ	1
407	CAPT	104	CO HQ	1
408	1LT	104	CO HQ	1
409	DRIVER	104	CO HQ	1
410	GUNNER	104	CO HQ	1
411	LOADER	104	CO HQ	1
412	PILOT	104	CO HQ	1
413	2LT	106	CMD SN	1
414	DRIVER	106	CMD SN	1
415	GUNNER	106	CMD SN	1
416	LOADER	106	CMD SN	1
417	CHIEF	107	ARMR SN	1

(Table 2 continued next page)

Table 2, continued.

418	DRIVER	107	ARMR SN	1
419	GUNNER	107	ARMR SN	1
420	LOADER	107	ARMR SN	1
801	M1A1	102	BN HQ	1
802	M23	102	BN HQ	1
803	M1R	102	BN HQ	1
804	M1A1	104	CO HQ	1
805	M23	104	CO HQ	1
806	M1R	104	CO HQ	1
807	M1A1	106	CMD SN	1
808	M23	106	CMD SN	1
809	M1R	106	CMD SN	1
810	M1A1	107	ARMR SN	1
811	M23	107	ARMR SN	1
812	M1R	107	ARMR SN	1
1201	UH1V	102	BN HQ	1
1202	UH1V	104	CO HQ	1

**Table-3: Characteristic**

Characteristic RDS ID	Characteristic SDS ID	Characteristic Name
201	Null	COUNTRY-CODE
601	Null	PERSONNEL-QUANTITY-REQUIRED
701	Null	PERSONNEL-QUANTITY-AUTHORIZED
1001	Null	EQUIPMENT-QUANTITY-REQUIRED
1101	Null	EQUIPMENT-QUANTITY-AUTHORIZED
1401	Null	AIRCRAFT-QUANTITY-REQUIRED
1501	Null	AIRCRAFT-QUANTITY-AUTHORIZED

**Table-4: Entity-Characteristic**

<b>Entity RDS ID</b>	<b>Entity SDS ID</b>	<b>Characteristic RDS ID</b>	<b>Characteristic SDS ID</b>	<b>Entity Characteristic Numeric Value</b>	<b>Entity Characteristic String Value</b>
102	BN HQ	201	null	0	US
103	ARMR CO	201	null	0	US
104	CO HQ	201	null	0	US
105	ARMR PLT	201	null	0	US
106	CMD SN	201	null	0	US
107	ARMR SN	201	null	0	US
401	LT COL	601	null	1	
402	MAJ	601	null	1	
403	DRIVER	601	null	2	
404	GUNNER	601	null	2	
405	LOADER	601	null	2	
406	PILOT	601	null	2	
407	CAPT	601	null	1	
408	1LT	601	null	1	
409	DRIVER	601	null	2	
410	GUNNER	601	null	2	
411	LOADER	601	null	2	
412	PILOT	601	null	2	
413	2LT	601	null	1	
414	DRIVER	601	null	1	
415	GUNNER	601	null	1	
416	LOADER	601	null	1	
417	CHIEF	601	null	1	
418	DRIVER	601	null	1	
419	GUNNER	601	null	1	
420	LOADER	601	null	1	
401	LT COL	701	null	1	
402	MAJ	701	null	1	

Table 4(a), Continued from Table 4:

403	DRIVER	701	null	2	
404	GUNNER	701	null	2	
405	LOADER	701	null	2	
406	PILOT	701	null	2	
407	CAPT	701	null	1	
408	1LT	701	null	1	
409	DRIVER	701	null	2	
410	GUNNER	701	null	2	
411	LOADER	701	null	2	
412	PILOT	701	null	2	
413	2LT	701	null	1	
414	DRIVER	701	null	1	
415	GUNNER	701	null	1	
416	LOADER	701	null	1	
417	CHIEF	701	null	1	
418	DRIVER	701	null	1	
419	GUNNER	701	null	1	
420	LOADER	701	null	1	
801	M1A1	1001	null	2	
802	M23	1001	null	2	
803	M1R	1001	null	6	
804	M1A1	1001	null	2	
805	M23	1001	null	2	
806	M1R	1001	null	6	
807	M1A1	1001	null	1	
808	M23	1001	null	1	
809	M1R	1001	null	3	
810	M1A1	1001	null	1	
811	M23	1001	null	1	
812	M1R	1001	null	3	
801	M1A1	1101	null	2	
802	M23	1101	null	2	
803	M1R	1101	null	6	
804	M1A1	1101	null	2	
805	M23	1101	null	2	
806	M1R	1101	null	6	
807	M1A1	1101	null	1	
808	M23	1101	null	1	
809	M1R	1101	null	3	
810	M1A1	1101	null	1	
811	M23	1101	null	1	
812	M1R	1101	null	3	
1201	UH1V	1401	null	1	
1202	UH1V	1401	null	1	
1201	UH1V	1501	null	1	
1202	UH1V	1501	null	1	

## APPENDIX C

### CODE

```
package uob;

import java.io.Serializable;

/**
 * This class contains definition of the 'change' object
 * sent from the UOB system to the FDMS system
 */
public class ChangeObj implements Serializable
{
    public String operation;
    public String table;
    public Object obj;

    public ChangeObj(String operation, String table, Object obj)
    {
        this.operation = operation;
        this.table = table;
        this.obj = obj;
    }
};
```

```
package uob;

import java.io.Serializable;

/**
 * @author Jasmine
 *
 * Definition of the Unit Class
 */
public class Unit implements Serializable
{
    /**
     First item in class definition should be
     primary key in database for this class
     */
    public String pkey;
    public String unitId;
    public String parentUnitId;
    public String unitName;
    public String homeName;
    public String shipCategory;
    public String countryCode;
}
```

```
package uob;

import java.io.Serializable;

/**
 * @author Jasmine
 *
 * Definition of the Unit Aircraft Class
 */
public class UnitAircraft implements Serializable
{
    /**
     First item in class definition should be
     primary key in database for this class
     */
    public String pkey;
    public String aIC;
    public String unitId;
    public String aircraftCode;
    public String aircraftDesc;
    public String aircraftQtyReqd;
    public String aircraftQtyAuth;
}
```

```
package uob;

import java.io.Serializable;

/**
 * @author Jasmine
 *
 * Definition of the Unit Equipment Class
 */
public class UnitEquipment implements Serializable
{
    /**
     First item in class definition should be
     primary key in database for this class
     */
    public String pkey;
    public String eIC;
    public String unitId;
    public String equipCode;
    public String equipDesc;
    public String equipQtyReqd;
    public String equipQtyAuth;
}
```

```
// UnitPersonnel.java
```

```
package uob;
```

```
import java.io.Serializable;
```

```
/**
```

```
 * @author Jasmine
```

```
 *
```

```
 * Definition of the Unit Personnel Class
```

```
 */
```

```
public class UnitPersonnel implements Serializable
```

```
{
```

```
    /**
```

```
     First item in class definition should be
```

```
     primary key in database for this class
```

```
     */
```

```
     public String pkey;
```

```
     public String pIC;
```

```
     public String unitId;
```

```
     public String personnelDesc;
```

```
     public String personnelQtyReqd;
```

```
     public String personnelQtyAuth;
```

```
}
```

```
package uob;

import javax.ejb.EJBObject;
import java.rmi.RemoteException;

//import syncobj.ChangeObj;

/**
This interface defines the 'Remote' interface for the 'UobEJB' EJB. Its
methods are the only methods exposed to the outside world. The class
UobEJBBean implements these methods.
*/

public interface UobEJB extends EJBObject
{
    public void updateUOB( Unit obj ) throws RemoteException;
    public void updateUOB( UnitAircraft obj ) throws RemoteException;
    public void updateUOB( UnitEquipment obj ) throws RemoteException;
    public void updateUOB( UnitPersonnel obj ) throws RemoteException;

    public void createUOB( Unit obj ) throws RemoteException;
    public void createUOB( UnitAircraft obj ) throws RemoteException;
    public void createUOB( UnitEquipment obj ) throws RemoteException;
    public void createUOB( UnitPersonnel obj ) throws RemoteException;

    public void deleteUOB( Unit obj ) throws RemoteException;
    public void deleteUOB( UnitAircraft obj ) throws RemoteException;
    public void deleteUOB( UnitEquipment obj ) throws RemoteException;
    public void deleteUOB( UnitPersonnel obj ) throws RemoteException;

    public Object[] viewObjects( String className ) throws RemoteException;
    public boolean validateUser( String userName, String password ) throws
RemoteException;
}
```

```

// UobEJB.java

package uob;

import java.rmi.RemoteException;
import javax.ejb.SessionBean;
import javax.ejb.SessionContext;

import javax.naming.Context;
import javax.naming.InitialContext;
import javax.naming.NamingException;

import javax.jms.ObjectMessage;
import javax.jms.TopicConnectionFactory;
import javax.jms.TopicConnection;
import javax.jms.TopicSession;
import javax.jms.TopicPublisher;
import javax.jms.Topic;
import javax.jms.TextMessage;
import javax.jms.Session;
import javax.jms.JMSEXception;

import java.text.*;
import java.sql.*;
import javax.naming.*;

/**
This class contains implementation methods for the UobEJB session bean
*/

public class UobEJBBean implements SessionBean
{
    // Database information
    private Context dbCtx = null;
    private java.sql.Connection dbCon;

    // Updates a Unit table record with the record in 'obj'
    public void updateUOB( Unit obj )
    {
        // SQL helpers...
        String headStr = "update UNIT ";
        String tailStr = " where PKey like '" + obj.pkey + "'";
        String updateStr;
        try

```

```

{
    // Connect to the database, create a statement
    dbInit();
    Statement statement = dbCon.createStatement();

    // Execute SQL statements to update the database
    updateStr = headStr + "SET Unit_Identification_Code = " + obj.unitId + "" +
tailStr;
    statement.executeUpdate(updateStr);
    updateStr = headStr + "SET Parent_Unit_Id_Code = " + obj.parentUnitId + "" +
tailStr;
    statement.executeUpdate(updateStr);
    updateStr = headStr + "SET Unit_Name = " + obj.unitName + "" + tailStr;
    statement.executeUpdate(updateStr);
    updateStr = headStr + "SET Home_Name = " + obj.homeName + "" + tailStr;
    statement.executeUpdate(updateStr);
    updateStr = headStr + "SET Ship_Category = " + obj.shipCategory + "" + tailStr;
    statement.executeUpdate(updateStr);
    updateStr = headStr + "SET Country_Code = " + obj.countryCode + "" + tailStr;
    statement.executeUpdate(updateStr);

    // Close the database connection
    statement.close();
    dbClose();

}
// Trap Errors, if any
catch ( SQLException e )
{
    e.printStackTrace();
}

// Update FDMS if required
    updateFDMS ("Unit", "update", obj);

}

// Updates a Unit Aircraft table record with the record in 'obj'
public void updateUOB( UnitAircraft obj )
{
    // SQL helpers...
    String headStr = "update UNIT_AIRCRAFT ";
    String tailStr = " where PKey like " + obj.pkey + "";
    String updateStr;
    try

```

```

{
    // Connect to the database, create a statement
    dbInit();
    Statement statement = dbCon.createStatement();

    // Execute SQL statements to update the database
    updateStr = headStr + "SET AIC = " + obj.aIC + "" + tailStr;
    statement.executeUpdate(updateStr);
    updateStr = headStr + "SET Unit_Identification_Code = " + obj.unitId + "" +
tailStr;
    statement.executeUpdate(updateStr);
    updateStr = headStr + "SET Aircraft_Code = " + obj.aircraftCode + "" + tailStr;
    statement.executeUpdate(updateStr);
    updateStr = headStr + "SET Aircraft_Description = " + obj.aircraftDesc + "" +
tailStr;
    statement.executeUpdate(updateStr);
    updateStr = headStr + "SET Aircraft_Quantity_Required = " +
obj.aircraftQtyReqd + "" + tailStr;
    statement.executeUpdate(updateStr);
    updateStr = headStr + "SET Aircraft_Quantity_Authorized = " +
obj.aircraftQtyAuth + "" + tailStr;
    statement.executeUpdate(updateStr);
    //System.out.println(updateStr);

    // Close the database connection
    statement.close();
    dbClose();

}
// Trap Errors, if any
catch ( SQLException e )
{
    e.printStackTrace();
}

// Update FDMS if required
    updateFDMS ("Aircraft", "update", obj);

}

// Updates a Unit Equipment table record with the record in 'obj'
public void updateUOB( UnitEquipment obj )
{
    // SQL helpers...
    String headStr = "update UNIT_EQUIPMENT ";

```

```

        String tailStr = " where PKey like '" + obj.pkey + "'";
        String updateStr;
        try
    {
        // Connect to the database, create a statement
        dbInit();
        Statement statement = dbCon.createStatement();

        // Execute SQL statements to update the database
        updateStr = headStr + "SET EIC = '" + obj.eIC + "'" + tailStr;
        statement.executeUpdate(updateStr);
        updateStr = headStr + "SET Unit_Identification_Code = '" + obj.unitId + "'" +
tailStr;
        statement.executeUpdate(updateStr);
        updateStr = headStr + "SET Equipment_Code = '" + obj.equipCode + "'" + tailStr;
        statement.executeUpdate(updateStr);
        updateStr = headStr + "SET Equipment_Description = '" + obj.equipDesc + "'" +
tailStr;
        statement.executeUpdate(updateStr);
        updateStr = headStr + "SET Equipment_Quantity_Required = '" +
obj.equipQtyReqd + "'" + tailStr;
        statement.executeUpdate(updateStr);
        updateStr = headStr + "SET Equipment_Quantity_Authorized = '" +
obj.equipQtyAuth + "'" + tailStr;
        statement.executeUpdate(updateStr);

        // Close the database connection
        statement.close();
        dbClose();

    }
    // Trap Errors, if any
    catch ( SQLException e )
    {
        e.printStackTrace();
    }

    // Update FDMS if required
        updateFDMS ("Equipment", "update", obj);

    }

// Updates a Unit Personnel table record with the record in 'obj'
public void updateUOB( UnitPersonnel obj )
{

```

```

        // SQL helpers...
        String headStr = "update UNIT_PERSONNEL ";
        String tailStr = " where PKey like '" + obj.pkey + "'";
        String updateStr;
        try
    {
        // Connect to the database, create a statement
        dbInit();
        Statement statement = dbCon.createStatement();

        // Execute SQL statements to update the database
        updateStr = headStr + "SET PIC = '" + obj.pIC + "'" + tailStr;
        statement.executeUpdate(updateStr);
        updateStr = headStr + "SET Unit_Identification_Code = '" + obj.unitId + "'" +
tailStr;
        statement.executeUpdate(updateStr);
        updateStr = headStr + "SET Personnel_Description = '" + obj.personnelDesc + "'"
+ tailStr;
        statement.executeUpdate(updateStr);
        updateStr = headStr + "SET Personnel_Quantity_Required = '" +
obj.personnelQtyReqd + "'" + tailStr;
        statement.executeUpdate(updateStr);
        updateStr = headStr + "SET Personnel_Quantity_Authorized = '" +
obj.personnelQtyAuth + "'" + tailStr;
        statement.executeUpdate(updateStr);

        // Close the database connection
        statement.close();
        dbClose();

    }
    // Trap Errors, if any
    catch ( SQLException e )
    {
        e.printStackTrace();
    }

    // Update FDMS if required
        updateFDMS ("Personnel", "update", obj);

    }

    // Creates a new record in the Unit table using the record in 'obj'
    public void createUOB( Unit obj )
    {

```

```

// SQL helpers...
String updateStr = "INSERT into UNIT VALUES (";

// add in the database...
try
{
    dbInit();
    Statement statement = dbCon.createStatement();

    updateStr = updateStr + "" + obj.pkey + "" + "," + " ";
    updateStr = updateStr + "" + obj.unitId + "" + "," + " ";
    updateStr = updateStr + "" + obj.parentUnitId + "" + "," + " ";
    updateStr = updateStr + "" + obj.unitName + "" + "," + " ";
    updateStr = updateStr + "" + obj.homeName + "" + "," + " ";
    updateStr = updateStr + "" + obj.shipCategory + "" + "," + " ";
    updateStr = updateStr + "" + obj.countryCode + "" + ")";

    statement.executeUpdate(updateStr);
    statement.close();
    dbClose();

}
catch ( SQLException e )
{
    e.printStackTrace();
}

// Update FDMS if required
    updateFDMS ("Unit", "create", obj);
}

// Creates a new record in the Unit Aircraft table using the record in 'obj'
public void createUOB( UnitAircraft obj )
{
    dbInit();
    Statement statement;
    String queryString;
    ResultSet rs;

    // Make sure it's not a duplicate entry...
    // Get Aircraft Description from Unit_Aircraft table to test it
    queryString = "Select Aircraft_Description From Unit_Aircraft";
    queryString += " where AIC like '" + obj.aIC + "'";

```

```

queryString += " and Unit_Identification_Code like '" + obj.unitId + "'";

System.out.println(queryString);
try
{
    statement = dbCon.createStatement();
    rs = statement.executeQuery(queryString);
    rs.next();
    // If there is no exception, this is a duplicate.
    String airDesc = rs.getString(1);
    System.out.println("Duplicate Rejected");
    return;
}
catch (SQLException e)
{
    System.out.println("Entry not found in Aircraft table");
}

// Add new entry to Aircraft Table
String updateStr = "INSERT into UNIT_AIRCRAFT VALUES (";

try
{
    updateStr = updateStr + "" + obj.pkey + "" + "," + " ";
    updateStr = updateStr + "" + obj.aIC + "" + "," + " ";
    updateStr = updateStr + "" + obj.unitId + "" + "," + " ";
    updateStr = updateStr + "" + obj.aircraftCode + "" + "," + " ";
    updateStr = updateStr + "" + obj.aircraftDesc + "" + "," + " ";
    updateStr = updateStr + "" + obj.aircraftQtyReqd + "" + "," + " ";
    updateStr = updateStr + "" + obj.aircraftQtyAuth + "" + ")";

    statement = dbCon.createStatement();
    statement.executeUpdate(updateStr);
    statement.close();
    dbClose();
}
catch ( SQLException e )
{
    e.printStackTrace();
}

// Update FDMS if required
updateFDMS ("Aircraft", "create", obj);
}

```

```

// Creates a new record in the Unit Equipment table using the record in 'obj'
public void createUOB( UnitEquipment obj )
{
    dbInit();
    Statement statement;
    String queryString;
    ResultSet rs;

    // Make sure it's not a duplicate entry...
    // Get Equipment Description from Unit_Equipment table to test it
    queryString = "Select Equipment_Description From Unit_Equipment";
    queryString += " where EIC like '" + obj.eIC + "'";
    queryString += " and Unit_Identification_Code like '" + obj.unitId + "'";

    System.out.println(queryString);
    try
    {
        statement = dbCon.createStatement();
        rs = statement.executeQuery(queryString);
        rs.next();
        // If there is no exception, this is a duplicate.
        String equipDesc = rs.getString(1);
        System.out.println("Duplicate Rejected");
        return;
    }
    catch (SQLException e)
    {
        System.out.println("Entry not found in Equipment table");
    }

    // Add new entry to Equipment Table
    String updateStr = "INSERT into UNIT_EQUIPMENT VALUES (";

    try
    {
        updateStr = updateStr + "'" + obj.pkey + "'" + "," + " ";
        updateStr = updateStr + "'" + obj.eIC + "'" + "," + " ";
        updateStr = updateStr + "'" + obj.unitId + "'" + "," + " ";
        updateStr = updateStr + "'" + obj.equipCode + "'" + "," + " ";
        updateStr = updateStr + "'" + obj.equipDesc + "'" + "," + " ";
        updateStr = updateStr + "'" + obj.equipQtyReqd + "'" + "," + " ";
        updateStr = updateStr + "'" + obj.equipQtyAuth + "'" + ")";

        statement = dbCon.createStatement();
        statement.executeUpdate(updateStr);
        statement.close();
    }
}

```

```

        dbClose();
    }
    catch ( SQLException e )
    {
        e.printStackTrace();
    }

// Update FDMS if required
    updateFDMS ("Equipment", "create", obj);
}

// Creates a new record in the Unit Personnel table using the record in 'obj'
public void createUOB( UnitPersonnel obj )
{

    dbInit();
    Statement statement;
    String queryString;
    ResultSet rs;

    // Make sure it's not a duplicate entry...
    // Get Personnel Description from Unit_Personnel table to test it
    queryString = "Select Personnel_Description From Unit_Personnel";
    queryString += " where PIC like '" + obj.pIC + "'";
    queryString += " and Unit_Identification_Code like '" + obj.unitId + "'";

    System.out.println(queryString);
    try
    {
        statement = dbCon.createStatement();
        rs = statement.executeQuery(queryString);
        rs.next();
        // If there is no exception, this is a duplicate.
        String persDesc = rs.getString(1);
        System.out.println("Duplicate Rejected");
        return;
    }
    catch (SQLException e)
    {
        System.out.println("Entry not found in Personnel table");
    }

// Add new entry to Personnel Table

```

```

String updateStr = "INSERT into UNIT_PERSONNEL VALUES (";

    try
    {
        updateStr = updateStr + "" + obj.pkey + "" + "," + " ";
        updateStr = updateStr + "" + obj.pIC + "" + "," + " ";
        updateStr = updateStr + "" + obj.unitId + "" + "," + " ";
        updateStr = updateStr + "" + obj.personnelDesc + "" + "," + " ";
        updateStr = updateStr + "" + obj.personnelQtyReqd + "" + "," + " ";
        updateStr = updateStr + "" + obj.personnelQtyAuth + "" + ")";

        statement = dbCon.createStatement();
        statement.executeUpdate(updateStr);
        statement.close();
        dbClose();

    }
    catch ( SQLException e )
    {
        e.printStackTrace();
    }

// Update FDMS if required
    updateFDMS ("Personnel", "create", obj);
}

// Deletes a record in the Unit table whose copy is in 'obj'
public void deleteUOB( Unit obj )
{
    // SQL helpers...
    String updateStr = "DELETE from UNIT where PKey like "" + obj.pkey +
    """;

    // perform the delete in the database
    try
    {
        dbInit();
        Statement statement = dbCon.createStatement();
        statement.executeUpdate(updateStr);
        statement.close();
        dbClose();

    }
    catch ( SQLException e )
    {

```

```

    e.printStackTrace();
}

// Update FDMS if required
    updateFDMS ("Unit", "delete", obj);
}

// Deletes a record in the Unit Aircraft table whose copy is in 'obj'
public void deleteUOB( UnitAircraft obj )
{
    // SQL helpers...
    String updateStr = "DELETE from UNIT_AIRCRAFT where PKey like "
+ obj.pkey + """;

    // perform the delete in the database
    try
    {
        dbInit();
        Statement statement = dbCon.createStatement();
        statement.executeUpdate(updateStr);
        statement.close();
        dbClose();
    }
    catch ( SQLException e )
    {
        e.printStackTrace();
    }

// Update FDMS if required
    updateFDMS ("Aircraft", "delete", obj);
}

// Deletes a record in the Unit Equipment table whose copy is in 'obj'
public void deleteUOB( UnitEquipment obj )
{
    // SQL helpers...
    String updateStr = "DELETE from UNIT_EQUIPMENT where PKey like
"" + obj.pkey + """;

    // perform the delete in the database
    try
    {

```

```

        dbInit();
        Statement statement = dbCon.createStatement();
        statement.executeUpdate(updateStr);
        statement.close();
        dbClose();

    }
    catch ( SQLException e )
    {
        e.printStackTrace();
    }

    // Update FDMS if required
        updateFDMS ("Equipment", "delete", obj);
    }

    // Deletes a record in the Unit Personnel table whose copy is in 'obj'
    public void deleteUOB( UnitPersonnel obj )
    {
        // SQL helpers...
        String updateStr = "DELETE from UNIT_PERSONNEL where PKey like
"" + obj.pkey + """;
        // perform the delete in the database
        try
        {
            dbInit();
            Statement statement = dbCon.createStatement();
            statement.executeUpdate(updateStr);
            statement.close();
            dbClose();
        }
        catch ( SQLException e )
        {
            e.printStackTrace();
        }

        // Update FDMS if required
            updateFDMS ("Personnel", "delete", obj);
        }

    // Returns all records of the table in the database

```

```

// whose name is in the variable className
public Object[] viewObjects( String className )
{
    // Check the name of the table whose records are requested
    // Call the appropriate method that reads them from the database
    if (className.equals (Unit.class.toString()))
    {
        return readUnitData();
    }
    else if (className.equals (UnitAircraft.class.toString()))
    {
        return readUnitAircraftData();
    }
    else if (className.equals (UnitEquipment.class.toString()))
    {
        return readUnitEquipmentData();
    }
    else if (className.equals (UnitPersonnel.class.toString()))
    {
        return readUnitPersonnelData();
    }
    else
        return (Object[]) null;
}

```

```

// Returns all records in the Unit table
private Object[] readUnitData()
{
    try
    {
        //System.out.println("readUnitData");
        // get Unit data
        dbInit();
        Statement statement = dbCon.createStatement();

        // Get the count
        String queryString = "SELECT Count(*) FROM UNIT";
        ResultSet rs = statement.executeQuery(queryString);
        rs.next();
        int count = rs.getInt(1);
        Object[] objArray = new Object[count];
        //System.out.println("Count is: "+ count);

        // get all the objects
        queryString = "SELECT * FROM Unit";
    }
}

```

```

rs = statement.executeQuery(queryString);
int i = 0;

// copy all the objects in an array
while ((rs.next()) && (i < count))
{
    Unit tempUnit = new Unit();

    tempUnit.pkey = Integer.toString(rs.getInt(1));
    tempUnit.unitId = rs.getString(2);
    tempUnit.parentUnitId = rs.getString(3);
    tempUnit.unitName = rs.getString(4);
    tempUnit.homeName = rs.getString(5);
    tempUnit.shipCategory = rs.getString(6);
    tempUnit.countryCode = rs.getString(7);

    //System.out.println("pkey: " + tempUnit.pkey);
    //System.out.println("unitId: " + tempUnit.unitId);
    //System.out.println("parentUnitId: " +
tempUnit.parentUnitId);
    //System.out.println("unitName: " + tempUnit.unitName);
    //System.out.println("homeName: " +
tempUnit.homeName);
    //System.out.println("shipCategory: " +
tempUnit.shipCategory);
    //System.out.println("countryCode: " +
tempUnit.countryCode);

    objArray[i++] = tempUnit;
}

// close the database connection
rs.close();
statement.close();
dbClose();

// return all the objects in the array
return objArray;
}
catch (SQLException e)
{
    e.printStackTrace();
    return null;
}
}

```

```
// Returns all records in the Unit Aircraft table
private Object[] readUnitAircraftData()
{
    try
    {
        // get Unit data
        dbInit();
        Statement statement = dbCon.createStatement();

        // Get the count
        String queryString = "SELECT Count(*) FROM Unit_Aircraft";
        ResultSet rs = statement.executeQuery(queryString);
        rs.next();
        int count = rs.getInt(1);
        Object[] objArray = new Object[count];

        // get the objects
        queryString = "SELECT * FROM Unit_Aircraft";
        rs = statement.executeQuery(queryString);
        int i = 0;

        // copy all the objects in an array
        while ((rs.next()) && (i < count))
        {
            UnitAircraft tempUnit = new UnitAircraft();

            tempUnit.pkey = rs.getString(1);
            tempUnit.aIC = rs.getString(2);
            tempUnit.unitId = rs.getString(3);
            tempUnit.aircraftCode = rs.getString(4);
            tempUnit.aircraftDesc = rs.getString(5);
            tempUnit.aircraftQtyReqd = rs.getString(6);
            tempUnit.aircraftQtyAuth = rs.getString(7);

            objArray[i++] = tempUnit;
        }

        // close the database connection
        rs.close();
        statement.close();
        dbClose();

        // return all the objects in the array
        return objArray;
    }
}
```

```

        catch (SQLException e)
        {
            e.printStackTrace();
            return null;
        }
    }

    // Returns all records in the Unit Equipment table
    private Object[] readUnitEquipmentData()
    {
        try
        {
            // get Unit data
            dbInit();
            Statement statement = dbCon.createStatement();

            // Get the count
            String queryString = "SELECT Count(*) FROM
Unit_Equipment";
            ResultSet rs = statement.executeQuery(queryString);
            rs.next();
            int count = rs.getInt(1);
            Object[] objArray = new Object[count];

            // get the objects
            queryString = "SELECT * FROM Unit_Equipment";
            rs = statement.executeQuery(queryString);
            int i = 0;

            // copy all the objects in an array
            while ((rs.next()) && (i < count))
            {
                UnitEquipment tempUnit = new UnitEquipment();

                tempUnit.pkey = rs.getString(1);
                tempUnit.eIC = rs.getString(2);
                tempUnit.unitId = rs.getString(3);
                tempUnit.equipCode = rs.getString(4);
                tempUnit.equipDesc = rs.getString(5);
                tempUnit.equipQtyReqd = rs.getString(6);
                tempUnit.equipQtyAuth = rs.getString(7);

                objArray[i++] = tempUnit;
            }
        }
    }

```

```

        // close the database connection
        rs.close();
        statement.close();
        dbClose();

        // return all the objects in the array
        return objArray;
    }
    catch (SQLException e)
    {
        e.printStackTrace();
        return null;
    }
}

// Returns all records in the Unit Personnel table
private Object[] readUnitPersonnelData()
{
    try
    {
        // get Unit data
        dbInit();
        Statement statement = dbCon.createStatement();

        // Get the count
        String queryString = "SELECT Count(*) FROM Unit_Personnel";
        ResultSet rs = statement.executeQuery(queryString);
        rs.next();
        int count = rs.getInt(1);
        Object[] objArray = new Object[count];

        // get all the objects
        queryString = "SELECT * FROM Unit_Personnel";
        rs = statement.executeQuery(queryString);
        int i = 0;

        // copy them in an array that will be returned
        while ((rs.next()) && (i < count))
        {
            UnitPersonnel tempUnit = new UnitPersonnel();

            tempUnit.pkey = rs.getString(1);
            tempUnit.pIC = rs.getString(2);
            tempUnit.unitId = rs.getString(3);
            tempUnit.personnelDesc = rs.getString(4);
        }
    }
}

```

```

        tempUnit.personnelQtyReqd = rs.getString(5);
        tempUnit.personnelQtyAuth = rs.getString(6);

        objArray[i++] = tempUnit;
    }

    // close the database connection
    rs.close();
    statement.close();
    dbClose();

    // return all the objects in the array
    return objArray;
}
catch (SQLException e)
{
    e.printStackTrace();
    return null;
}
}

```

```

// Sends an 'change' packet to the FDMS database if the change is mapped
// in the FDMS database
private void updateFDMS (String tableName, String operation, Object obj)
{

    TopicConnection topicConnection;
    TopicSession topicSession;
    TopicPublisher topicPublisher;
    Topic topic;
    ChangeObj changeObj;

    // Check the mapping table and continue only if needed
    if ( (checkMappingTable(tableName)) != true )
        return;

    // Set up the change packet
    changeObj = new ChangeObj(tableName, operation, obj);

    /**
     * Name used to lookup TopicConnectionFactory
     */
    //final String CONNECTION_JNDI =
    "java:comp/env/jms/MyTopicConnection";
}

```

```

/**
 * Name used to lookup topic destination
 */
//final String TOPIC_JNDI = "java:comp/env/jms/TopicName";

final String CONNECTION_JNDI = "ConnectionFactory";
final String TOPIC_JNDI = "topic/testTopic";

try
{

    // Get the initial context
    Context context = new InitialContext();

    // Get the connection factory
    TopicConnectionFactory topicFactory =
        (TopicConnectionFactory)
context.lookup(CONNECTION_JNDI);

    // Create the connection
    topicConnection = topicFactory.createTopicConnection();

    // Create the session
    topicSession = topicConnection.createTopicSession(
        // No transaction
        false,
        // Auto ack
        Session.AUTO_ACKNOWLEDGE);

    // Look up the destination
    topic = (Topic) context.lookup(TOPIC_JNDI);

    // Create a publisher
    topicPublisher = topicSession.createPublisher(topic);

    // Create a message
    ObjectMessage objectMsg = topicSession.createObjectMessage();
    objectMsg.setObject(changeObj);

    // Publish the message
    topicPublisher.publish(objectMsg);

    //close connections..
    topicSession.close();
}

```

```

        topicConnection.close();
    }
    catch (Exception ex)
    {
        ex.printStackTrace();
    }
}

```

```

private boolean checkMappingTable (String tableName)
{
    // The current table design of the UOB maps all its entries
    // to the FDMS database. Hence, all changes should be sent
    // to the FDMS system

    return true;
}

```

```

// Validates username and password entered by a user for logging in
public boolean validateUser(String userName, String password)
{
    try
    {
        String pw = " ";
        dbInit();
        Statement statement = dbCon.createStatement();
        String queryString =
            "SELECT password FROM users WHERE name='" +
userName + "'";

        ResultSet rs = statement.executeQuery(queryString);
        while (rs.next())
        {
            pw = rs.getString("password");
        }

        rs.close();
        statement.close();
        dbClose();

        // if invalid password, return
        if (pw.equals(password))

```

```

        return true;
    else
        return false;
    }
    catch (SQLException e)
    {
        e.printStackTrace();
        return false;
    }
}

/**
 * Create database connections
 */
private void dbInit()
{
    try
    {
        dbCtx = new InitialContext();

        javax.sql.DataSource ds = (javax.sql.DataSource) dbCtx.lookup
("java:MySqlDS");

        //String url =
"jdbc:mysql://127.0.0.1/jdbctest?user=tester&password=tester";
        dbCon = ds.getConnection("developer", "mysql");

    }
    catch (Exception e)
    {
        // a failure occurred
        try
        {
            if (dbCon != null)
            {
                dbCon.close();
            }
            if (dbCtx!=null)
            {
                dbCtx.close();
            }
            e.printStackTrace();
        }
    }
}

```

```
        catch (SQLException SQLe)
        {
        }
        catch (NamingException SQLe)
        {
        }
    }
}
```

```
/**
Do closing housekeeping
*/
private void dbClose()
{
    try
    {
        if (dbCon != null)
        {
            dbCon.close();
        }
        if (dbCtx!=null)
        {
            dbCtx.close();
        }
    }
    catch (SQLException SQLe)
    {
        SQLe.printStackTrace();
    }
    catch (NamingException SQLe)
    {
        SQLe.printStackTrace();
    }
}
```

```
/**
Empty method body
*/
public void ejbCreate() {}
```

```
/**
```

```
Empty method body
*/
public void ejbRemove() {}

/**
Empty method body
*/
public UobEJBBean() {}

/**
Empty method body
*/
public void ejbActivate() {}

/**
Empty method body
*/
public void ejbPassivate() {}

/**
Empty method body
*/
public void setSessionContext(SessionContext sc) {}
}
```

```
// UnitPersonnel.java

package uob;

import java.io.Serializable;

/**
 * @author Jasmine
 *
 * Definition of the Unit Personnel Class
 */
public class UnitPersonnel implements Serializable
{
    /**
     First item in class definition should be
     primary key in database for this class
     */
    public String pkey;
    public String pIC;
    public String unitId;
    public String personnelDesc;
    public String personnelQtyReqd;
    public String personnelQtyAuth;
}
```

```
//  
  
package uob;  
  
import java.io.Serializable;  
import java.rmi.RemoteException;  
import javax.ejb.CreateException;  
import javax.ejb.EJBHome;  
  
/**  
This interface defines the `home' interface for the `UobEJB' EJB.  
*/  
  
public interface UobEJBHome extends EJBHome  
{  
    /**  
        Creates an instance of the `UobEJBBean' class on the server, and returns a  
        remote reference to an UobEJB interface on the client.  
    */  
    UobEJB create() throws RemoteException, CreateException;  
}
```

```
//  
  
package fdms;  
  
import java.io.Serializable;  
  
/**  
    This class contains definition of the 'change' object  
    sent from the UOB system to the FDMS system  
*/  
  
public class ChangeObj implements Serializable  
{  
    public String operation;  
    public String table;  
    public Object obj;  
  
    public ChangeObj(String operation, String table, Object obj)  
    {  
        this.operation = operation;  
        this.table = table;  
        this.obj = obj;  
    }  
};
```

```
//  
  
package fdms;  
  
import java.io.Serializable;  
  
/**  
 * @author Jasmine  
 *  
 * Definition of the Unit Class  
 */  
public class Unit implements Serializable  
{  
    /**  
     First item in class definition should be  
     primary key in database for this class  
     */  
    public String pkey;  
    public String unitId;  
    public String parentUnitId;  
    public String unitName;  
    public String homeName;  
    public String shipCategory;  
    public String countryCode;  
}
```



```

    <p align="center">Unit Name</p>
  </td>
</tr>
<tr>
  <td width="40%" align="center">
    <p align="center"><html:text property="homename" size="32"/></p>
  </td>
  <td width="60%" align="center">
    <p align="center">Home Name</p>
  </td>
</tr>
<tr>
  <td width="40%" align="center">
    <p align="center"><html:text property="shipcategory" size="32"/></p>
  </td>
  <td width="60%" align="center">
    <p align="center">Ship Category</p>
  </td>
</tr>
<tr>
  <td width="40%" align="center">
    <p align="center"><html:text property="countrycode" size="32"/></p>
  </td>
  <td width="60%" align="center">
    <p align="center">Country Code</p>
  </td>
</tr>
</table>

<BR><BR><center> <html:submit property="ok" value="Ok"/> </center>

</html:form>

</body>

</html:html>

```

```
//createUnitAircraft
```

```
<%@ taglib uri="/WEB-INF/struts-html.tld" prefix="html" %>
```

```
<%-- This JSP creates a form for a user to enter values for a new record in the Unit Aircraft table --%>
```

```
<html:html>
```

```
<head>
```

```
<title>Edit Unit Aircraft</title>
```

```
</head>
```

```
<body bgcolor="#7D7DE6">
```

```
<html:errors/>
```

```
<p align="center"><font size="4"><b><u>UOB Editor</u></b></font></p>
```

```
<p align="center" style="margin-bottom: 30"><b>Create Unit Aircraft</b></p>
```

```
<p align="center" style="margin-bottom: 30"><b>Unit Aircraft</b></p>
```

```
<html:form action="CreateUnitA.do">
```

```
<table border="1" width="29%" align="center">
```

```
<tr>
```

```
<td width="40%" align="center">
```

```
<p align="center"><html:text property="aic" size="32"/></p>
```

```
</td>
```

```
<td width="60%" align="center">
```

```
<p align="center">AIC&nbsp;&nbsp;&nbsp;</p>
```

```
</td>
```

```
</tr>
```

```
<tr>
```

```
<td width="40%" align="center">
```

```
<p align="center"><html:text property="unitid" size="32"/></p>
```

```
</td>
```

```
<td width="60%" align="center">
```

```
<p align="center">Unit Id&nbsp;&nbsp;&nbsp;</p>
```

```
</td>
```

```
</tr>
```

```
<tr>
```

```
<td width="40%" align="center">
```

```
<p align="center"><html:text property="aircraftcode" size="32"/></p>
```

```
</td>
```

```
<td width="60%" align="center">
```

```

    <p align="center">Aircraft Code</p>
  </td>
</tr>
<tr>
  <td width="40%" align="center">
    <p align="center"><html:text property="aircraftdesc" size="32"/></p>
  </td>
  <td width="60%" align="center">
    <p align="center">Aircraft Desc</p>
  </td>
</tr>
<tr>
  <td width="40%" align="center">
    <p align="center"><html:text property="aircraftqtyreqd" size="32"/></p>
  </td>
  <td width="60%" align="center">
    <p align="center">Aircraft Qty Reqd</p>
  </td>
</tr>
<tr>
  <td width="40%" align="center">
    <p align="center"><html:text property="aircraftqtyauth" size="32"/></p>
  </td>
  <td width="60%" align="center">
    <p align="center">Aircraft Qty Auth</p>
  </td>
</tr>
</table>

<BR><BR><center> <html:submit property="ok" value="Ok"/> </center>

</html:form>

</body>

</html:html>

```

```
//createUnitEquipment

<%@ taglib uri="/WEB-INF/struts-html.tld" prefix="html" %>

<%-- This JSP creates a form for a user to enter values for a new record in the Unit
Equipment table --%>

<html:html>

<head>
  <title>Edit Unit Equipment</title>
</head>

<body bgcolor="#7D7DE6">

<html:errors/>

<p align="center"><font size="4"><b><u>UOB Editor</u></b></font></p>
<p align="center" style="margin-bottom: 30"><b>Create Unit Equipment</b></p>
<p align="center" style="margin-bottom: 30"><b>Unit Equipment</b></p>

<html:form action="CreateUnitE.do">

<table border="1" width="29%" align="center">
  <tr>
    <td width="40%" align="center">
      <p align="center"><html:text property="eic" size="32"/></p>
    </td>
    <td width="60%" align="center">
      <p align="center">EIC&nbsp;&nbsp;&nbsp;</p>
    </td>
  </tr>
  <tr>
    <td width="40%" align="center">
      <p align="center"><html:text property="unitid" size="32"/></p>
    </td>
    <td width="60%" align="center">
      <p align="center">Unit Id&nbsp;&nbsp;&nbsp;</p>
    </td>
  </tr>
  <tr>
    <td width="40%" align="center">
      <p align="center"><html:text property="equipmentcode" size="32"/></p>
    </td>
    <td width="60%" align="center">
```

```

    <p align="center">Equipment Code</p>
  </td>
</tr>
<tr>
  <td width="40%" align="center">
    <p align="center"><html:text property="equipmentdesc" size="32"/></p>
  </td>
  <td width="60%" align="center">
    <p align="center">Equipment Desc</p>
  </td>
</tr>
<tr>
  <td width="40%" align="center">
    <p align="center"><html:text property="equipmentqtyreqd" size="32"/></p>
  </td>
  <td width="60%" align="center">
    <p align="center">Equipment Qty Reqd</p>
  </td>
</tr>
<tr>
  <td width="40%" align="center">
    <p align="center"><html:text property="equipmentqtyauth" size="32"/></p>
  </td>
  <td width="60%" align="center">
    <p align="center">Equipment Qty Auth</p>
  </td>
</tr>
</table>

<BR><BR><center> <html:submit property="ok" value="Ok"/> </center>

</html:form>

</body>

</html:html>

```



```
<td width="60%" align="center">
  <p align="center">Personnel Desc</p>
</td>
</tr>
<tr>
  <td width="40%" align="center">
    <p align="center"><html:text property="personnelqtyreqd" size="32"/></p>
  </td>
  <td width="60%" align="center">
    <p align="center">Personnel Qty Reqd</p>
  </td>
</tr>
<tr>
  <td width="40%" align="center">
    <p align="center"><html:text property="personnelqtyauth" size="32"/></p>
  </td>
  <td width="60%" align="center">
    <p align="center">Personnel Qty Auth</p>
  </td>
</tr>
</table>

<BR><BR><center> <html:submit property="ok" value="Ok"/> </center>

</html:form>

</body>

</html:html>
```

```
//editUnit.jsp
```

```
<%@ taglib uri="/WEB-INF/struts-html.tld" prefix="html" %>
```

```
<%-- This JSP creates a form for a user to edit values of a record in the Unit table --%>
```

```
<html:html>
```

```
<head>
```

```
<title>Edit Unit</title>
```

```
</head>
```

```
<body bgcolor="#7D7DE6">
```

```
<html:errors/>
```

```
<p align="center"><font size="4"><b><u>UOB Editor</u></b></font></p>
```

```
<p align="center" style="margin-bottom: 30"><b>Edit Unit</b></p>
```

```
<p align="center" style="margin-bottom: 30"><b>Unit</b></p>
```

```
<html:form action="EditUnit.do">
```

```
<table border="1" width="29%" align="center">
```

```
<tr>
```

```
<td width="40%" align="center">
```

```
<p align="center"><html:text property="unitid" size="32"/></p>
```

```
</td>
```

```
<td width="60%" align="center">
```

```
<p align="center">Unit Id&nbsp;&nbsp;&nbsp;</p>
```

```
</td>
```

```
</tr>
```

```
<tr>
```

```
<td width="40%" align="center">
```

```
<p align="center"><html:text property="parentunitid" size="32"/></p>
```

```
</td>
```

```
<td width="60%" align="center">
```

```
<p align="center">Parent Unit Id&nbsp;&nbsp;&nbsp;</p>
```

```
</td>
```

```
</tr>
```

```
<tr>
```

```
<td width="40%" align="center">
```

```
<p align="center"><html:text property="unitname" size="32"/></p>
```

```
</td>
```

```
<td width="60%" align="center">
```

```

    <p align="center">Unit Name</p>
  </td>
</tr>
<tr>
  <td width="40%" align="center">
    <p align="center"><html:text property="homename" size="32"/></p>
  </td>
  <td width="60%" align="center">
    <p align="center">Home Name</p>
  </td>
</tr>
<tr>
  <td width="40%" align="center">
    <p align="center"><html:text property="shipcategory" size="32"/></p>
  </td>
  <td width="60%" align="center">
    <p align="center">Ship Category</p>
  </td>
</tr>
<tr>
  <td width="40%" align="center">
    <p align="center"><html:text property="countrycode" size="32"/></p>
  </td>
  <td width="60%" align="center">
    <p align="center">Country Code</p>
  </td>
</tr>
</table>

<BR><BR><center> <html:submit property="ok" value="Ok"/> </center>

</html:form>

</body>

</html:html>

```

```
//editUnitAircraft.jsp
```



```

</td>
</tr>
<tr>
  <td width="40%" align="center">
    <p align="center"><html:text property="aircraftdesc" size="32"/></p>
  </td>
  <td width="60%" align="center">
    <p align="center">Aircraft Desc</p>
  </td>
</tr>
<tr>
  <td width="40%" align="center">
    <p align="center"><html:text property="aircraftqtyreqd" size="32"/></p>
  </td>
  <td width="60%" align="center">
    <p align="center">Aircraft Qty Reqd</p>
  </td>
</tr>
<tr>
  <td width="40%" align="center">
    <p align="center"><html:text property="aircraftqtyauth" size="32"/></p>
  </td>
  <td width="60%" align="center">
    <p align="center">Aircraft Qty Auth</p>
  </td>
</tr>
</table>

<BR><BR><center> <html:submit property="ok" value="Ok"/> </center>

</html:form>

</body>

</html:html>

```



```

    <p align="center">Equipment Code</p>
  </td>
</tr>
<tr>
  <td width="40%" align="center">
    <p align="center"><html:text property="equipmentdesc" size="32"/></p>
  </td>
  <td width="60%" align="center">
    <p align="center">Equipment Desc</p>
  </td>
</tr>
<tr>
  <td width="40%" align="center">
    <p align="center"><html:text property="equipmentqtyreqd" size="32"/></p>
  </td>
  <td width="60%" align="center">
    <p align="center">Equipment Qty Reqd</p>
  </td>
</tr>
<tr>
  <td width="40%" align="center">
    <p align="center"><html:text property="equipmentqtyauth" size="32"/></p>
  </td>
  <td width="60%" align="center">
    <p align="center">Equipment Qty Auth</p>
  </td>
</tr>
</table>

<BR><BR><center> <html:submit property="ok" value="Ok"/> </center>

</html:form>

</body>

</html:html>

```

```
//editUnitPersonnel
```



```
</td>
</tr>
<tr>
  <td width="40%" align="center">
    <p align="center"><html:text property="personnelqtyreqd" size="32"/></p>
  </td>
  <td width="60%" align="center">
    <p align="center">Personnel Qty Reqd</p>
  </td>
</tr>
<tr>
  <td width="40%" align="center">
    <p align="center"><html:text property="personnelqtyauth" size="32"/></p>
  </td>
  <td width="60%" align="center">
    <p align="center">Personnel Qty Auth</p>
  </td>
</tr>
</table>

<BR><BR><center> <html:submit property="ok" value="Ok"/> </center>

</html:form>

</body>

</html:html>
```

```
<%-- This JSP informs a user about login errors --%>
```

```
<html>
```

```
  <head>
```

```
    <title>
```

```
      UOB Editor: Error!!
```

```
    </title>
```

```
  </head>
```

```
  <body bgcolor="#7D7DE6">
```

```
    <BR>
```

```
    <center>
```

```
      <h2>UOB Data Editor: Login Error!!</h2>
```

```
    </center>
```

```
    <BR>
```

```
    <center>
```

```
      <B>Login activity had error(s): </B>
```

```
      <P>
```

```
      <B>Please try again... </B>
```

```
      <P><BR>
```

```
      <form action="/DbUobWT/form/login.jsp">
```

```
        <input type="submit" name = "ok" value="OK">
```

```
      </form>
```

```
    </center>
```

```
</body>
```

```
</HTML>
```

```
//login.jsp
```

```
<%@ taglib uri="/WEB-INF/struts-html.tld" prefix="html"%>
```

```
<%-- This JSP creates a login form where a user can enter a username and password --
%>
```

```
<html>
```

```
    <head>
```

```
        <title>
```

```
            UOB Data Editor Login
```

```
        </title>
```

```
    </head>
```

```
    <body bgcolor="#7D7DE6">
```

```
        <html:errors/>
```

```
        <html:form action="/login">
```

```
            <BR>
```

```
            <center>
```

```
                <h2>UOB Editor: Login</h2>
```

```
            </center>
```

```
            <BR>
```

```
        <center>
```

```
            <table border=1 cellpadding=4 cellspacing=8>
```

```
        <!-- input name -->
```

```
            <tr bgcolor=#9D9DE6>
```

```
                <td>User Name</td><td><html:text
```

```
property="name"/></td><td><html:errors property="name"/></td>
```

```
            </tr>
```

```
        <!-- input password -->
```

```
            <tr bgcolor=#9D9DE6>
```

```
                <td>Password</td><td><html:password
```

```
property="password"/></td><td><html:errors property="password"/></td>
```

```
            </tr>
```

```
</table>
</center>

<BR><BR>
<center><html:submit property="ok" value="Login"/></center>
```

```
        </html:form>
    </body>
</html>
```

```
//uobView.jsp
```

```
<%@ page language="java" %>
```

```
<%@ taglib uri="/WEB-INF/struts-bean.tld" prefix="bean" %>
```

```
<%@ taglib uri="/WEB-INF/struts-html.tld" prefix="html" %>
```

```
<%@ taglib uri="/WEB-INF/struts-logic.tld" prefix="logic" %>
```

```
<%-- This JSP displays all records of all the UOB tables --%>
```

```
<html:html>
```

```
<head>
```

```
  <title>
```

```
    UOB Data Editor: View Tables
```

```
  </title>
```

```
</head>
```

```
<script>
```

```
function go(action)
```

```
{
```

```
  document.forms[0].action.value=action;
```

```
}
```









</p>

```
<table border="1" width="100%">
```

```
<tr>
```

```
<td width="19%" align="center"><b>PIC</b></td>
```

```
<td width="19%" align="center"><b>Unit Identification Code</b></td>
```

```
<td width="23%" align="center"><b>Personnel Description</b></td>
```

```
<td width="23%" align="center"><b>Personnel Qty Required</b></td>
```

```
<td width="23%" align="center"><b>Personnel Qty Auth</b></td>
```

```
<td width="8%" align="center"><b>Select</b></td>
```

```
</tr>
```

```
<%
```

```
UnitPersonnel unitP = new UnitPersonnel();
```

```
readObj = uobEditor.viewObjects(unitP.getClass().toString());
```

```
//session.setAttribute("sizeOfUnitP", Integer.toString(readObj.length));
```

```
unitPStart = unitEEnd;
```

```
unitPEnd = unitPStart + readObj.length;
```

```
session.setAttribute("unitPStart", Integer.toString(unitPStart));
```

```
session.setAttribute("unitPEnd", Integer.toString(unitPEnd));
```

```
// logout value is same as P End
```

```
session.setAttribute("logoutValue", Integer.toString(unitPEnd));
```

```
for (int i=0; i < readObj.length; i++)
```

```
{
```

```
    unitP = (UnitPersonnel)readObj[i];
```

```
%>
```

```
<tr>
```

```
<td width="19%" align="center"> <%= unitP.pIC%> </td>
```

```
<td width="19%" align="center"> <%= unitP.unitId%> </td>
```

```
<td width="23%" align="center"> <%= unitP.personnelDesc%> </td>
```

```
<td width="23%" align="center"> <%= unitP.personnelQtyReqd%> </td>
```

```
<td width="23%" align="center"> <%= unitP.personnelQtyAuth%> </td>
```

```
<td width="8%" align="center"> <html:radio property="optionSelect" value="<%=
```

```
String.valueOf(kk++) %>"/></td>
```

```
</tr>
```

```
<%
```

```
}
```

```
%>
```

```
</table>
```

```
<p><br></p>
```

```
</html:form>
```

```
</body>
```

```
</html:html>
```

```
//createUnitAction.jsp
```

```
package uobwt;
```

```
import java.io.IOException;
```

```
import javax.servlet.*;
```

```
import javax.servlet.http.*;
```

```
import org.apache.struts.action.Action;
```

```
import org.apache.struts.action.ActionForm;
```

```
import org.apache.struts.action.ActionForward;
```

```
import org.apache.struts.action.ActionMapping;
```

```
import uob.*;
```

```
/**
```

```
 * @author jasmine
```

```
 *
```

```
 * This servlet class creates a new record in the Unit table
```

```
 */
```

```
public class CreateUnitAction extends Action
```

```
{
```

```
    public ActionForward execute(  
        ActionMapping mapping,  
        ActionForm form,  
        HttpServletRequest request,  
        HttpServletResponse response)  
        throws Exception
```

```
{
```

```
//System.out.println("CreateUnitAction");

// Setup the associated form
EditunitForm unitForm = (EditunitForm) form;
// Get the session variable for the application tier session bean
HttpSession session = request.getSession();
UobEJB uobEditor = (UobEJB)session.getAttribute("uobEditor");

// If didn't come here the right way, generate error!!
if (uobEditor == null)
{
    return (mapping.findForward("error"));
}

// Create a new Unit (record) object and copy form values in it
Unit unit = new Unit();

unit.pkey = "0";
unit.unitId = unitForm.unitid;
unit.parentUnitId = unitForm.parentunitid;
    unit.unitName = unitForm.unitname;
    unit.homeName = unitForm.homename;
    unit.shipCategory = unitForm.shipcategory;
    unit.countryCode = unitForm.countrycode;

// do the update
uobEditor.createUOB(unit);

// return success to controlling servlet
// That means, display the view window next
return mapping.findForward("view");
}
}
```

```

//createUnitAircraft.java

package uobwt;

import java.io.IOException;

import javax.servlet.*;
import javax.servlet.http.*;

import org.apache.struts.action.Action;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;

import uob.*;

/**
 * @author jasmine
 *
 * This servlet class creates a new record in the Unit Aircraft table
 */
public class CreateUnitAircraftAction extends Action
{

    public ActionForward execute(
        ActionMapping mapping,
        ActionForm form,
        HttpServletRequest request,
        HttpServletResponse response)
        throws Exception
    {
        //System.out.println("CreateUnitAircraftAction");

        // Setup the associated form
        EditunitAircraftForm unitaForm = (EditunitAircraftForm) form;
        // Get the session variable for the application tier session bean
        HttpSession session = request.getSession();
        UobEJB uobEditor = (UobEJB)session.getAttribute("uobEditor");

        // If didn't come here the right way, generate error!!
        if (uobEditor == null)
        {
            return (mapping.findForward("error"));
        }
    }
}

```

```
// Create a new Unit Aircraft (record) object and copy form values in it
UnitAircraft unitA = new UnitAircraft();
```

```
unitA.pkey = "0";
unitA.aIC = unitaForm.aic;
unitA.unitId = unitaForm.unitid;
unitA.aircraftCode = unitaForm.aircraftcode;
unitA.aircraftDesc = unitaForm.aircraftdesc;
unitA.aircraftQtyReqd = unitaForm.aircraftqtyreqd;
unitA.aircraftQtyAuth = unitaForm.aircraftqtyauth;
```

```
//System.out.println("pkey; " + unitA.pkey);
//System.out.println("aircraftQtyAuth; " + unitA.aircraftQtyAuth);
```

```
// do the update
uobEditor.createUOB(unitA);
```

```
// return success to controlling servlet
// That means, display the view window next
return mapping.findForward("view");
```

```
}
```

```
}
```

```
//createUnitEquipment.java

package uobwt;

import java.io.IOException;

import javax.servlet.*;
import javax.servlet.http.*;

import org.apache.struts.action.Action;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;

import uob.*;

/**
 * @author jasmine
 *
 * This servlet class creates a new record in the Unit Equipment table
 */
public class CreateUnitEquipmentAction extends Action
{

    public ActionForward execute(
        ActionMapping mapping,
        ActionForm form,
        HttpServletRequest request,
        HttpServletResponse response)
        throws Exception
    {
        //System.out.println("CreateUnitEAction");

        // Setup the associated form
        EditunitEquipmentForm uniteForm = (EditunitEquipmentForm) form;
        // Get the session variable for the application tier session bean
        HttpSession session = request.getSession();
        UobEJB uobEditor = (UobEJB)session.getAttribute("uobEditor");

        // If didn't come here the right way, generate error!!
        if (uobEditor == null)
        {
            return (mapping.findForward("error"));
        }
    }
}

```

```

    }

    // Create a new Unit Equipment (record) object and copy form values in it
    UnitEquipment unitE = new UnitEquipment();

    unitE.pkey = "0";
    unitE.eIC = uniteForm.eic;
    unitE.unitId = uniteForm.unitid;
    unitE.equipCode = uniteForm.equipmentcode;
    unitE.equipDesc = uniteForm.equipmentdesc;
    unitE.equipQtyReqd = uniteForm.equipmentqtyreqd;
    unitE.equipQtyAuth = uniteForm.equipmentqtyauth;

    // do the update
    uobEditor.createUOB(unitE);

    // return success to controlling servlet
    // That means, display the view window next
    return mapping.findForward("view");
}
}

```

```
//createUnitPersonnel.java
```

```
package uobwt;
```

```
import java.io.IOException;
```

```
import javax.servlet.*;
```

```
import javax.servlet.http.*;
```

```
import org.apache.struts.action.Action;
```

```
import org.apache.struts.action.ActionForm;
```

```
import org.apache.struts.action.ActionForward;
```

```
import org.apache.struts.action.ActionMapping;
```

```
import uob.*;
```

```
/**
```

```

* @author jasmine
*
* This servlet class creates a new record in the Unit Personnel table
*/
public class CreateUnitPersonnelAction extends Action
{
    public ActionForward execute(
        ActionMapping mapping,
        ActionForm form,
        HttpServletRequest request,
        HttpServletResponse response)
        throws Exception
    {
        //System.out.println("CreateUnitPersonnelAction");

        // Setup the associated form
        EditunitPersonnelForm unitpForm = (EditunitPersonnelForm) form;
        // Get the session variable for the application tier session bean
        HttpSession session = request.getSession();
        UobEJB uobEditor = (UobEJB)session.getAttribute("uobEditor");

        // If didn't come here the right way, generate error!!
        if (uobEditor == null)
        {
            return (mapping.findForward("error"));
        }

        // Create a new Unit Personnel (record) object and copy form values in it
        UnitPersonnel unitP = new UnitPersonnel();

        unitP.pkey = "0";
        unitP.pIC = unitpForm.pic;
        unitP.unitId = unitpForm.unitid;
        unitP.personnelDesc = unitpForm.personneldesc;
        unitP.personnelQtyReqd = unitpForm.personnelqtyreqd;
        unitP.personnelQtyAuth = unitpForm.personnelqtyauth;

        // do the update
        uobEditor.createUOB(unitP);

        // return success to controlling servlet
        // That means, display the view window next
        return mapping.findForward("view");
    }
}

```

```
//DeleteAction.java

package uobwt;

import java.io.IOException;

import javax.servlet.*;
import javax.servlet.http.*;

import org.apache.struts.action.Action;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;

import uob.*;

/**
 * @author jasmine
 *
 * This servlet class deletes a record in the specified table
 */
public class DeleteAction extends Action
{

    /**
     * @see org.apache.struts.action.Action#execute(ActionMapping, ActionForm,
     HttpServletRequest, HttpServletResponse, )
     */
    public ActionForward execute(
        ActionMapping mapping,
        ActionForm form,
        HttpServletRequest request,
        HttpServletResponse response)
        throws Exception
    {
        //System.out.println("DeleteAction");
    }
}
```

```

        // Setup the associated form
        ViewForm welcomeForm = (ViewForm) form;
        // Get the session variable for the application tier session bean
        HttpSession session = request.getSession();
        UobEJB uobEditor = (UobEJB)session.getAttribute("uobEditor");

        // If didn't come here the right way, generate error!!
        if (uobEditor == null)
        {
            return (mapping.findForward("error"));
        }

        // Get the number of the selection that the user made
        String optionSelect = welcomeForm.getOptionSelect();
        //System.out.println("OptionValue: " + optionSelect);
        int optionValue=0;
        try
        {
            optionValue = Integer.parseInt(optionSelect);
        }
        catch (NumberFormatException e)
        {
            // Ignore wrong input and return to the 'uobview' page
            return (mapping.findForward("view"));
            //e.printStackTrace();
        }

        // Get the boundaries for the records in each table
        int unitStart = Integer.parseInt((String)session.getAttribute("unitStart"));
        int unitEnd = Integer.parseInt((String)session.getAttribute("unitEnd"));

        int unitAStart =
Integer.parseInt((String)session.getAttribute("unitAStart"));
        int unitAEnd = Integer.parseInt((String)session.getAttribute("unitAEnd"));

        int unitEStart =
Integer.parseInt((String)session.getAttribute("unitEStart"));
        int unitEEnd = Integer.parseInt((String)session.getAttribute("unitEEnd"));

        int unitPStart =
Integer.parseInt((String)session.getAttribute("unitPStart"));
        int unitPEnd = Integer.parseInt((String)session.getAttribute("unitPEnd"));

        // If a unit record, delete it

```

```

if ((optionValue >= unitStart) && (optionValue < unitEnd))
{
    // find the record
    Unit unit = new Unit();
    Object[] readObj =
uobEditor.viewObjects(unit.getClass().toString());
    unit = (Unit)readObj[optionValue-unitStart];

    // delete the record
    uobEditor.deleteUOB(unit);

    // return to view screen
    return (mapping.findForward("view"));
}

// Else, if a unit aircraft record, delete it
else if ( (optionValue >= unitAStart) && (optionValue < unitAEnd))
{
    // find the record
    UnitAircraft unita = new UnitAircraft();
    Object[] readObj =
uobEditor.viewObjects(unita.getClass().toString());
    unita = (UnitAircraft)readObj[optionValue-unitAStart];

    // delete the record
    uobEditor.deleteUOB(unita);

    // return to view screen
    return (mapping.findForward("view"));
}

// Else, if a unit equipment record, delete it
else if ( (optionValue >= unitEStart) && (optionValue < unitEEnd))
{
    // find the record
    UnitEquipment unite = new UnitEquipment();
    Object[] readObj =
uobEditor.viewObjects(unite.getClass().toString());
    //System.out.println("optionvalue: " + optionValue);
    unite = (UnitEquipment)readObj[optionValue-unitEStart];

    // delete the record
    uobEditor.deleteUOB(unite);

    // return to view screen
    return (mapping.findForward("view"));
}

```

```

// Else, if a unit personnel record, delete it
else if ( (optionValue >= unitPStart) && (optionValue < unitPEnd))
{
    // find the record
    UnitPersonnel unitp = new UnitPersonnel();
    Object[] readObj =
uobEditor.viewObjects(unitp.getClass().toString());
    unitp = (UnitPersonnel)readObj[optionValue-unitPStart];

    // delete the record
    uobEditor.deleteUOB(unitp);

    // return to view screen
    return (mapping.findForward("view"));
}
// return error if the selection value is invalid
else return (mapping.findForward("error"));
}
}

```

```
//EditAction.java
```

```
package uobwt;
```

```
import java.io.IOException;
```

```
import javax.servlet.*;
```

```
import javax.servlet.http.*;
```

```
import org.apache.struts.action.Action;
```

```
import org.apache.struts.action.ActionForm;
```

```
import org.apache.struts.action.ActionForward;
```

```
import org.apache.struts.action.ActionMapping;
```

```
import uob.*;
```

```
/**
```

```
 * @author jasmine
```

```
 *
```

```
 * This servlet class processes the editing of a record in the specified table
```

```

*/
public class EditAction extends Action
{
    /**
     * @see org.apache.struts.action.Action#execute(ActionMapping, ActionForm,
     * HttpServletRequest, HttpServletResponse, )
     */
    public ActionForward execute(
        ActionMapping mapping,
        ActionForm form,
        HttpServletRequest request,
        HttpServletResponse response)
        throws Exception
    {
        // Setup the associated form
        ViewForm welcomeForm = (ViewForm) form;
        // Get the session variable for the application tier session bean
        HttpSession session = request.getSession();
        UobEJB uobEditor = (UobEJB)session.getAttribute("uobEditor");

        // If didn't come here the right way, generate error!!
        if (uobEditor == null)
        {
            return (mapping.findForward("error"));
        }

        // Get the number of the selection that the user made
        String optionSelect = welcomeForm.getOptionSelect();
        //System.out.println("OptionValue: " + optionSelect);
        int optionValue=0;
        try
        {
            optionValue = Integer.parseInt(optionSelect);
        }
        catch (NumberFormatException e)
        {
            return (mapping.findForward("view"));
            //e.printStackTrace();
        }

        // Get the boundaries for the records in each table
        int unitStart = Integer.parseInt((String)session.getAttribute("unitStart"));
        int unitEnd = Integer.parseInt((String)session.getAttribute("unitEnd"));
    }
}

```

```

        int unitAStart =
Integer.parseInt((String)session.getAttribute("unitAStart"));
        int unitAEnd = Integer.parseInt((String)session.getAttribute("unitAEnd"));

        int unitEStart =
Integer.parseInt((String)session.getAttribute("unitEStart"));
        int unitEEnd = Integer.parseInt((String)session.getAttribute("unitEEnd"));

        int unitPStart =
Integer.parseInt((String)session.getAttribute("unitPStart"));
        int unitPEnd = Integer.parseInt((String)session.getAttribute("unitPEnd"));

        // Depending on the table, request the controlling servlet to display
        // the form for editing the appropriate record.

        // unit record
        if ((optionValue >= unitStart) && (optionValue < unitEnd))
        {
            session.setAttribute("EditValue", optionSelect);
            return (mapping.findForward("editunitsetup"));
        }
        // unit aircraft record
        else if ( (optionValue >= unitAStart) && (optionValue < unitAEnd))
        {
            session.setAttribute("EditValue", optionSelect);
            return (mapping.findForward("editunitasetup"));
        }
        // unit equipment record
        else if ( (optionValue >= unitEStart) && (optionValue < unitEEnd))
        {
            session.setAttribute("EditValue", optionSelect);
            return (mapping.findForward("editunitsetup"));
        }
        // unit personnel record
        else if ( (optionValue >= unitPStart) && (optionValue < unitPEnd))
        {
            session.setAttribute("EditValue", optionSelect);
            return (mapping.findForward("editunitpsetup"));
        }
        // return error if the selection value is invalid
        else return (mapping.findForward("error"));
    }
}

```

```
//EditunitAction.java
```

```

package uobwt;
import java.io.*;
import java.util.*;
import org.apache.struts.action.*;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;

import uob.*;

public class EditunitAction extends Action{

    /**
     * This servlet class processes the editing of a record in the Unit table
     */
    public ActionForward execute(ActionMapping mapping,
                                ActionForm form,
                                HttpServletRequest request,
                                HttpServletResponse response
                                ) throws Exception

    {
        // Setup the associated form
        EditunitForm unitForm = (EditunitForm) form;
        // Get the session variable for the application tier session bean
        HttpSession session = request.getSession();
        UobEJB uobEditor = (UobEJB)session.getAttribute("uobEditor");

        // If didn't come here the right way, generate error!!
        if (uobEditor == null)
        {
            return (mapping.findForward("error"));
        }

        // Create a new Unit (record) object and copy form values in it
        Unit unit = new Unit();

        unit.pkey = (String)session.getAttribute("currentPkey");
        unit.unitId = unitForm.unitid;
        unit.parentUnitId = unitForm.parentunitid;
        unit.unitName = unitForm.unitname;
        unit.homeName = unitForm.homename;
    }
}

```

```

        unit.shipCategory = unitForm.shipcategory;
        unit.countryCode = unitForm.countrycode;

        // do the update
        uobEditor.updateUOB(unit);

        // return success to controlling servlet
        // That means, display the view window next
        return mapping.findForward("view");
    }
} //End class

```

```
//EditunitActionSetup.java
```

```

package uobwt;
import java.io.*;
import java.util.*;
import org.apache.struts.action.*;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;

import uob.*;

/**
 * @author jasmine
 *
 * This servlet class is called after a user selects a Unit record
 * in the View window and clicks the Edit button of that table. It identifies
 * the record that the user would like to edit, reads it from the application-tier
 * session bean and loads the values of its columns in the associated form, to allow
 * the controlling servlet to display those values in the form displayed by
 * EditUnit.jsp. The user can then edit the displayed values
 */
public class EditunitActionSetup extends Action
{

```

```

public ActionForward execute(ActionMapping mapping,
                             ActionForm form,
                             HttpServletRequest request,
                             HttpServletResponse response
                             ) throws Exception

{
    //System.out.println("EditunitActionSetup");

    // Setup the associated form
    EditunitForm unitForm = (EditunitForm) form;

    // Get the session variable for the application tier session bean
    HttpSession session = request.getSession();
    UobEJB uobEditor = (UobEJB)session.getAttribute("uobEditor");

    // If didn't come here the right way, generate error!!
    if (uobEditor == null)
    {
        return (mapping.findForward("error"));
    }

    // get the record selection number that the user made
    String editValue = (String)session.getAttribute("EditValue");
    int unitStart = Integer.parseInt((String)session.getAttribute("unitStart"));
    int unitEnd = Integer.parseInt((String)session.getAttribute("unitEnd"));

    // make sure it's valid and within range of the values for the unit table
    if (editValue == null)
    {
        return (mapping.findForward("error"));
    }

    int optionValue=0;
    try
    {
        // make sure it's in range; else, return error
        optionValue = Integer.parseInt(editValue);
        if ((optionValue < unitStart) || (optionValue >= unitEnd ))
            return (mapping.findForward("error"));
        // adjust optionValue for local index
        optionValue = optionValue - unitStart;
    }
    catch (NumberFormatException e)
    {

```

```

        e.printStackTrace();
    }

    // create a new unit object
    Unit unit = new Unit();

    // read all the records in the unit table
    Object[] readObj = uobEditor.viewObjects(unit.getClass().toString());

    // using the selection made by thee user,
    // copy the record we want in the new object
    unit = (Unit)readObj[optionValue];

    // copy the values of the record we just read into the associated form
    session.setAttribute("currentPkey", unit.pkey );
    unitForm.pkey = unit.pkey;
    unitForm.unitid = unit.unitId;
    unitForm.parentunitid = unit.parentUnitId;
    unitForm.unitname = unit.unitName;
    unitForm.homename = unit.homeName;
    unitForm.shipcategory = unit.shipCategory;
    unitForm.countrycode = unit.countryCode;

    // return success to the controlling servlet
    // that means, display the Edit Unit form
    return mapping.findForward("editunit");
}
} //End class

```

```
// EditunitAircraftAction
```

```

package uobwt;
import java.io.*;
import java.util.*;
import org.apache.struts.action.*;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;
import uob.*;

```

```
/**
```

```
* @author jasmine
```

```

*
* This servlet class edits a record in the Unit Aircraft table
*/
public class EditunitAircraftAction extends Action
{

    public ActionForward execute(ActionMapping mapping,
                                ActionForm form,
                                HttpServletRequest request,
                                HttpServletResponse response
                                ) throws Exception

    {
        // Setup the associated form
        EditunitAircraftForm unitaForm = (EditunitAircraftForm) form;
        // Get the session variable for the application tier session bean
        HttpSession session = request.getSession();
        UobEJB uobEditor = (UobEJB)session.getAttribute("uobEditor");

        // If didn't come here the right way, generate error!!
        if (uobEditor == null)
        {
            return (mapping.findForward("error"));
        }

        // Create a new Unit Aircraft (record) object and copy form values in it
        UnitAircraft unitA = new UnitAircraft();

        unitA.pkey = (String)session.getAttribute("currentPkey");
        unitA.aIC = unitaForm.aic;
        unitA.unitId = unitaForm.unitid;
        unitA.aircraftCode = unitaForm.aircraftcode;
        unitA.aircraftDesc = unitaForm.aircraftdesc;
        unitA.aircraftQtyReqd = unitaForm.aircraftqtyreqd;
        unitA.aircraftQtyAuth = unitaForm.aircraftqtyauth;

        //System.out.println("pkey; " + unitA.pkey);
        //System.out.println("aircraftQtyAuth; " + unitA.aircraftQtyAuth);

        // do the update
        uobEditor.updateUOB(unitA);

        // return success to controlling servlet
        // That means, display the view window next
        return mapping.findForward("view");
    }
}

```



```

        HttpServletResponse response
        ) throws Exception

    {
        //System.out.println("EditunitAircraftActionSetup");

        // Setup the associated form
        EditunitAircraftForm unitaForm = (EditunitAircraftForm) form;

        // Get the session variable for the application tier session bean
        HttpSession session = request.getSession();
        UobEJB uobEditor = (UobEJB)session.getAttribute("uobEditor");

        // If didn't come here the right way, generate error!!
        if (uobEditor == null)
        {
            return (mapping.findForward("error"));
        }

        // get the record selection number that the user made
        String editValue = (String)session.getAttribute("EditValue");

        // make sure it's valid and within range of the values for the unit aircraft table
        int unitAStart = Integer.parseInt((String)session.getAttribute("unitAStart"));
        int unitAEnd = Integer.parseInt((String)session.getAttribute("unitAEnd"));
        if (editValue == null)
        {
            return (mapping.findForward("error"));
        }

        int optionValue=0;
        try
        {
            // make sure it's in range; else, return error
            optionValue = Integer.parseInt(editValue);
            if ((optionValue < unitAStart) || (optionValue >= unitAEnd ))
                return (mapping.findForward("error"));
            // adjust optionValue for local index
            optionValue = optionValue - unitAStart;
        }
        catch (NumberFormatException e)
        {
            e.printStackTrace();
        }

        // create a new unit aircraft object

```

```
UnitAircraft unitA = new UnitAircraft();

// read all the records in the unit aircraft table
Object[] readObj = uobEditor.viewObjects(unitA.getClass().toString());

// using the selection made by thee user,
// copy the record we want in the new object
unitA = (UnitAircraft)readObj[optionValue];

// copy the values of the record we just read into the associated form
session.setAttribute("currentPkey", unitA.pkey );
unitaForm.pkey = unitA.pkey;
unitaForm.aic = unitA.aIC;
unitaForm.unitid = unitA.unitId;
unitaForm.aircraftcode = unitA.aircraftCode;
unitaForm.aircraftdesc = unitA.aircraftDesc;
unitaForm.aircraftqtyreqd = unitA.aircraftQtyReqd;
unitaForm.aircraftqtyauth = unitA.aircraftQtyAuth;

//System.out.println("pkey: " + unitaForm.pkey);

// return success to the controlling servlet
// that means, display the Edit Unit Aircraft form
return mapping.findForward("editunitA");
    }
} //End class
```

```
//EditunitAircraftForm

/**
 * This class defines a form for storage of values of a Unit Aircraft record
 */

package uobwt;
import org.apache.struts.action.ActionForm;

public class EditunitAircraftForm extends ActionForm
{

    /**
     * Member variable declaration
     */

    protected String pkey;
    protected String aic;
    protected String unitid;
    protected String aircraftcode;
    protected String aircraftdesc;
    protected String aircraftqtyreqd;
    protected String aircraftqtyauth;
    protected String ok;

    /**
     * method to set the value
     * @param String
     * @return void
     */
    public void setAircraftdesc(String aircraftdesc)
    {
        this.aircraftdesc = aircraftdesc;
    } //End method setAircraftdesc

    /**
     * method to get the value
     * @return String
     *
     */
    public String getAircraftdesc()
    {
        return this.aircraftdesc;
    }
}
```

```
} //End method getAircraftdesc

/**
 * method to set the value
 * @param String
 * @return void
 */
public void setOk(String ok)
{
    this.ok = ok;
} //End method setOk

/**
 * method to get the value
 * @return String
 */
public String getOk()
{
    return this.ok;
} //End method getOk

/**
 * method to set the value
 * @param String
 * @return void
 */
public void setAircraftqtyauth(String aircraftqtyauth)
{
    this.aircraftqtyauth = aircraftqtyauth;
} //End method setAircraftqtyauth

/**
 * method to get the value
 * @return String
 */
public String getAircraftqtyauth()
{
    return this.aircraftqtyauth;
} //End method getAircraftqtyauth
```

```
/**
 * method to set the value
 * @param String
 * @return void
 */
public void setAircraftqtyreqd(String aircraftqtyreqd)
{
    this.aircraftqtyreqd = aircraftqtyreqd;
} //End method setAircraftqtyreqd
```

```
/**
 * method to get the value
 * @return String
 *
 */
public String getAircraftqtyreqd()
{
    return this.aircraftqtyreqd;
} //End method getAircraftqtyreqd
```

```
/**
 * method to set the value
 * @param String
 * @return void
 */
public void setUnitid(String unitid)
{
    this.unitid = unitid;
} //End method setUnitid
```

```
/**
 * method to get the value
 * @return String
 *
 */
public String getUnitid()
{
    return this.unitid;
} //End method getUnitid
```

```
/**
```

```
* method to set the value
* @param String
* @return void
*/
public void setAic(String aic)
{
    this.aic = aic;
} //End method setAic

/**
 * method to get the value
 * @return String
 *
 */
public String getAic()
{
    return this.aic;
} //End method getAic

/**
 * method to set the value
 * @param String
 * @return void
 */
public void setAircraftcode(String aircraftcode)
{
    this.aircraftcode = aircraftcode;
} //End method setAircraftcode

/**
 * method to get the value
 * @return String
 *
 */
public String getAircraftcode()
{
    return this.aircraftcode;
} //End method getAircraftcode

/**
 * method to set the value
 * @param String
```

```

    * @return void
    */
    public void setPkey(String pkey)
    {
        this.pkey = pkey;
    } //End method setPkey

    /**
     * method to get the value
     * @return String
     *
     */
    public String getPkey()
    {
        return this.pkey;
    } //End method getPkey

} //End class

```

```
//EditunitEquipmentAction.java
```

```

package uobwt;
import java.io.*;
import java.util.*;
import org.apache.struts.action.*;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;
import uob.*;

```

```

/**
 * @author jasmine
 *
 * This servlet class edits a record in the Unit Equipment table
 */

```

```

public class EditunitEquipmentAction extends Action
{

    public ActionForward execute(ActionMapping mapping,
                                ActionForm form,
                                HttpServletRequest request,
                                HttpServletResponse response
                                ) throws Exception

    {
        // Setup the associated form
        EditunitEquipmentForm uniteForm = (EditunitEquipmentForm) form;
        // Get the session variable for the application tier session bean
        HttpSession session = request.getSession();
        UobEJB uobEditor = (UobEJB)session.getAttribute("uobEditor");

        // If didn't come here the right way, generate error!!
        if (uobEditor == null)
        {
            return (mapping.findForward("error"));
        }

        // Create a new Unit Equipment (record) object and copy form values in it
        UnitEquipment unitE = new UnitEquipment();

        unitE.pkey = (String)session.getAttribute("currentPkey");
        unitE.eIC = uniteForm.eic;
        unitE.unitId = uniteForm.unitid;
        unitE.equipCode = uniteForm.equipmentcode;
        unitE.equipDesc = uniteForm.equipmentdesc;
        unitE.equipQtyReqd = uniteForm.equipmentqtyreqd;
        unitE.equipQtyAuth = uniteForm.equipmentqtyauth;

        // do the update
        uobEditor.updateUOB(unitE);

        // return success to controlling servlet
        // That means, display the view window next
        return mapping.findForward("view");

    }
} //End class

```

```
// EditunitEquipmentActionSetup.java

package uobwt;
import java.io.*;
import java.util.*;
import org.apache.struts.action.*;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;

import uob.*;

/**
 * @author jasmine
 *
 * This servlet class is called after a user selects a Unit Equipment record
 * in the View window and clicks the Edit button of that table. It identifies
 * the record that the user would like to edit, reads it from the application-tier
 * session bean and loads the values of its columns in the associated form, to allow
 * the controlling servlet to display those values in the form displayed by
 * EditUnitEquipment.jsp. The user can then edit the displayed values
 */
public class EditunitEquipmentActionSetup extends Action
{

    public ActionForward execute(ActionMapping mapping,
                                ActionForm form,
                                HttpServletRequest request,
                                HttpServletResponse response
                                ) throws Exception

    {
        //System.out.println("EditunitEquipmentActionSetup");

        // Setup the associated form
        EditunitEquipmentForm uniteForm = (EditunitEquipmentForm) form;

        // Get the session variable for the application tier session bean
        HttpSession session = request.getSession();
        UobEJB uobEditor = (UobEJB)session.getAttribute("uobEditor");

        // If didn't come here the right way, generate error!!
    }
}

```

```

if (uobEditor == null)
{
    return (mapping.findForward("error"));
}

// get the record selection number that the user made
String editValue = (String)session.getAttribute("EditValue");

// make sure it's valid and within range of the values for the unit equipment table
int unitEStart = Integer.parseInt((String)session.getAttribute("unitEStart"));
    int unitEEnd = Integer.parseInt((String)session.getAttribute("unitEEnd"));
if (editValue == null)
{
    return (mapping.findForward("error"));
}

int optionValue=0;
    try
    {
        // make sure it's in range; else, return error
        optionValue = Integer.parseInt(editValue);
        if ((optionValue < unitEStart) || (optionValue >= unitEEnd ))
            return (mapping.findForward("error"));
        // adjust optionValue for local index
        optionValue = optionValue - unitEStart;
    }
    catch (NumberFormatException e)
    {
        e.printStackTrace();
    }

// create a new unit equipment object
UnitEquipment unitE = new UnitEquipment();

// read all the records in the unit equipment table
Object[] readObj = uobEditor.viewObjects(unitE.getClass().toString());

// using the selection made by thee user,
// copy the record we want in the new object
unitE = (UnitEquipment)readObj[optionValue];

// copy the values of the record we just read into the associated form
session.setAttribute("currentPkey", unitE.pkey );
uniteForm.pkey = unitE.pkey;
uniteForm.eic = unitE.eIC;
uniteForm.unitid = unitE.unitId;

```

```
uniteForm.equipmentcode = unitE.equipCode;
uniteForm.equipmentdesc = unitE.equipDesc;
uniteForm.equipmentqtyreqd = unitE.equipQtyReqd;
uniteForm.equipmentqtyauth = unitE.equipQtyAuth;

// return success to the controlling servlet
// that means, display the Edit Unit Equipment form
return mapping.findForward("editunitE");
}
} //End class
```

```
// EditunitEquipmentForm.java

/**
 * This class defines a form for storage of values of a Unit Equipment record
 */

package uobwt;
import org.apache.struts.action.ActionForm;

public class EditunitEquipmentForm extends ActionForm
{

    /**
     * Member variable declaration
     */

    protected String pkey;
    protected String unitid;
    protected String equipmentcode;
    protected String equipmentdesc;
    protected String equipmentqtyreqd;
    protected String equipmentqtyauth;
    protected String eic;
    protected String ok;

    /**
     * method to set the value
     * @param String
     * @return void
     */
    public void setEquipmentqtyauth(String equipmentqtyauth)
    {
        this.equipmentqtyauth = equipmentqtyauth;
    } //End method setEquipmentqtyauth

    /**
     * method to get the value
     * @return String
     *
     */
    public String getEquipmentqtyauth()
```

```
{
    return this.equipmentqtyauth;
} //End method getEquipmentqtyauth

/**
 * method to set the value
 * @param String
 * @return void
 */
public void setEquipmentqtyreqd(String equipmentqtyreqd)
{
    this.equipmentqtyreqd = equipmentqtyreqd;
} //End method setEquipmentqtyreqd

/**
 * method to get the value
 * @return String
 *
 */
public String getEquipmentqtyreqd()
{
    return this.equipmentqtyreqd;
} //End method getEquipmentqtyreqd

/**
 * method to set the value
 * @param String
 * @return void
 */
public void setEquipmentcode(String equipmentcode)
{
    this.equipmentcode = equipmentcode;
} //End method setEquipmentcode

/**
 * method to get the value
 * @return String
 *
 */
public String getEquipmentcode()
{
    return this.equipmentcode;
}
```

```
} //End method getEquipmentcode

/**
 * method to set the value
 * @param String
 * @return void
 */
public void setOk(String ok)
{
    this.ok = ok;
} //End method setOk

/**
 * method to get the value
 * @return String
 */
public String getOk()
{
    return this.ok;
} //End method getOk

/**
 * method to set the value
 * @param String
 * @return void
 */
public void setEquipmentdesc(String equipmentdesc)
{
    this.equipmentdesc = equipmentdesc;
} //End method setEquipmentdesc

/**
 * method to get the value
 * @return String
 */
public String getEquipmentdesc()
{
    return this.equipmentdesc;
} //End method getEquipmentdesc
```

```
/**
 * method to set the value
 * @param String
 * @return void
 */
public void setUnitid(String unitid)
{
    this.unitid = unitid;
} //End method setUnitid
```

```
/**
 * method to get the value
 * @return String
 *
 */
public String getUnitid()
{
    return this.unitid;
} //End method getUnitid
```

```
/**
 * method to set the value
 * @param String
 * @return void
 */
public void setEic(String eic)
{
    this.eic = eic;
} //End method setEic
```

```
/**
 * method to get the value
 * @return String
 *
 */
public String getEic()
{
    return this.eic;
} //End method getEic
```

```
/**
 * method to set the value
```

```
* @param String
* @return void
*/
public void setPkey(String pkey)
{
    this.pkey = pkey;
} //End method setPkey

/**
* method to get the value
* @return String
*
*/
public String getPkey()
{
    return this.pkey;
} //End method getPkey

} //End class
```

```
//EditunitForm.java

/**
 * This class defines a form for storage of values of a Unit record
 */

package uobwt;
import org.apache.struts.action.ActionForm;

public class EditunitForm extends ActionForm
{

    /**
     * Member variable declaration
     */

    protected String pkey;
    protected String shipcategory;
    protected String countrycode;
    protected String unitname;
    protected String homename;
    protected String ok;
    protected String unitid;
    protected String parentunitid;

    /**
     * method to set the value
     * @param String
     * @return void
     */
    public void setShipcategory(String shipcategory)
    {
        this.shipcategory = shipcategory;
    } //End method setShipcategory

    /**
     * method to get the value
     * @return String
     *
     */
    public String getShipcategory()
    {
        return this.shipcategory;
    }
}
```

```
} //End method getShipcategory

/**
 * method to set the value
 * @param String
 * @return void
 */
public void setCountrycode(String countrycode)
{
    this.countrycode = countrycode;
} //End method setCountrycode

/**
 * method to get the value
 * @return String
 *
 */
public String getCountrycode()
{
    return this.countrycode;
} //End method getCountrycode

/**
 * method to set the value
 * @param String
 * @return void
 */
public void setUnitname(String unitname)
{
    this.unitname = unitname;
} //End method setUnitname

/**
 * method to get the value
 * @return String
 *
 */
public String getUnitname()
{
    return this.unitname;
} //End method getUnitname
```

```
/**
 * method to set the value
 * @param String
 * @return void
 */
public void setHomename(String homename)
{
    this.homename = homename;
} //End method setHomename
```

```
/**
 * method to get the value
 * @return String
 *
 */
public String getHomename()
{
    return this.homename;
} //End method getHomename
```

```
/**
 * method to set the value
 * @param String
 * @return void
 */
public void setOk(String ok)
{
    this.ok = ok;
} //End method setOk
```

```
/**
 * method to get the value
 * @return String
 *
 */
public String getOk()
{
    return this.ok;
} //End method getOk
```

```
/**
```

```
* method to set the value
* @param String
* @return void
*/
public void setUnitid(String unitid)
{
    this.unitid = unitid;
} //End method setUnitid

/**
 * method to get the value
 * @return String
 *
 */
public String getUnitid()
{
    return this.unitid;
} //End method getUnitid

/**
 * method to set the value
 * @param String
 * @return void
 */
public void setParentunitid(String parentunitid)
{
    this.parentunitid = parentunitid;
} //End method setParentunitid

/**
 * method to get the value
 * @return String
 *
 */
public String getParentunitid()
{
    return this.parentunitid;
} //End method getParentunitid

/**
 * method to set the value
 * @param String
 * @return void
```

```
    */
    public void setPkey(String pkey)
    {
        this.pkey = pkey;
    } //End method setPkey

    /**
     * method to get the value
     * @return String
     *
     */
    public String getPkey()
    {
        return this.pkey;
    } //End method getPkey

} //End class
```

```
//EditunitPersonnelAction.java
```

```
package uobwt;
import java.io.*;
import java.util.*;
import org.apache.struts.action.*;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;
import uob.*;
```

```
/**
```

```

* @author jasmine
*
* This servlet class edits a record in the Unit Personnel table
*/
public class EditunitPersonnelAction extends Action
{

    public ActionForward execute(ActionMapping mapping,
                                ActionForm form,
                                HttpServletRequest request,
                                HttpServletResponse response
                                ) throws Exception

    {
        // Setup the associated form
        EditunitPersonnelForm unitpForm = (EditunitPersonnelForm) form;
        // Get the session variable for the application tier session bean
        HttpSession session = request.getSession();
        UobEJB uobEditor = (UobEJB)session.getAttribute("uobEditor");

        // If didn't come here the right way, generate error!!
        if (uobEditor == null)
        {
            return (mapping.findForward("error"));
        }

        // Create a new Unit Personnel (record) object and copy form values in it
        UnitPersonnel unitP = new UnitPersonnel();

        unitP.pkey = (String)session.getAttribute("currentPkey");
        unitP.pIC = unitpForm.pic;
        unitP.unitId = unitpForm.unitid;
        unitP.personnelDesc = unitpForm.personnelDesc;
        unitP.personnelQtyReqd = unitpForm.personnelQtyReqd;
        unitP.personnelQtyAuth = unitpForm.personnelQtyAuth;

        // do the update
        uobEditor.updateUOB(unitP);

        // return success to controlling servlet
        // That means, display the view window next
        return mapping.findForward("view");
    }
} //End class

```

```
//EditunitPersonnelActionSetup.java

package uobwt;
import java.io.*;
import java.util.*;
import org.apache.struts.action.*;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;

import uob.*;

/**
 * @author jasmine
 *
 * This servlet class is called after a user selects a Unit Personnel record
 * in the View window and clicks the Edit button of that table. It identifies
 * the record that the user would like to edit, reads it from the application-tier
 * session bean and loads the values of its columns in the associated form, to allow
 * the controlling servlet to display those values in the form displayed by
 * EditUnitPersonnel.jsp. The user can then edit the displayed values
 */
public class EditunitPersonnelActionSetup extends Action
{

    public ActionForward execute(ActionMapping mapping,
                                ActionForm form,
                                HttpServletRequest request,
                                HttpServletResponse response
                                ) throws Exception

    {
        //System.out.println("EditunitPersonnelActionSetup");

        // Setup the associated form
        EditunitPersonnelForm unitpForm = (EditunitPersonnelForm) form;

        // Get the session variable for the application tier session bean
        HttpSession session = request.getSession();
        UobEJB uobEditor = (UobEJB)session.getAttribute("uobEditor");

        // If didn't come here the right way, generate error!!
    }
}

```

```

if (uobEditor == null)
{
    return (mapping.findForward("error"));
}

// get the record selection number that the user made
String editValue = (String)session.getAttribute("EditValue");

// make sure it's valid and within range of the values for the unit personnel table
int unitPStart = Integer.parseInt((String)session.getAttribute("unitPStart"));
    int unitPEnd = Integer.parseInt((String)session.getAttribute("unitPEnd"));
if (editValue == null)
{
    return (mapping.findForward("error"));
}

int optionValue=0;
try
{
    // make sure it's in range; else, return error
    optionValue = Integer.parseInt(editValue);
    if ((optionValue < unitPStart) || (optionValue >= unitPEnd ))
        return (mapping.findForward("error"));
    // adjust optionValue for local index
    optionValue = optionValue - unitPStart;
}
catch (NumberFormatException e)
{
    e.printStackTrace();
}

// create a new unit personnel object
UnitPersonnel unitP = new UnitPersonnel();

// read all the records in the unit personnel table
Object[] readObj = uobEditor.viewObjects(unitP.getClass().toString());

// using the selection made by thee user,
// copy the record we want in the new object
unitP = (UnitPersonnel)readObj[optionValue];

// copy the values of the record we just read into the associated form
session.setAttribute("currentPkey", unitP.pkey );
unitpForm.pkey = unitP.pkey;
unitpForm.pic = unitP.pic;
unitpForm.unitid = unitP.unitId;

```

```

        unitpForm.personneldesc = unitP.personnelDesc;
        unitpForm.personnelqtyreqd = unitP.personnelQtyReqd;
        unitpForm.personnelqtyauth = unitP.personnelQtyAuth;

        // return success to the controlling servlet
        // that means, display the Edit Unit Personnel form
        return mapping.findForward("editunitP");
    }
} //End class

```

```
//EditunitPersonnelForm.java
```

```

/**
 * This class defines a form for storage of values of a Unit Personnel record
 */

```

```

package uobwt;
import org.apache.struts.action.ActionForm;

```

```

public class EditunitPersonnelForm extends ActionForm
{

```

```

    /**
     * Member variable declaration
     */

```

```

        protected String pkey;
        protected String personnelqtyauth;
        protected String personnelqtyreqd;
        protected String ok;
        protected String pic;
        protected String personneldesc;
        protected String unitid;

```

```

    /**
     * method to set the value
     * @param String
     * @return void

```

```
*/  
public void setPersonnelqtyauth(String personnelqtyauth)  
{  
    this.personnelqtyauth = personnelqtyauth;  
} //End method setPersonnelqtyauth
```

```
/**  
 * method to get the value  
 * @return String  
 *  
 */  
public String getPersonnelqtyauth()  
{  
    return this.personnelqtyauth;  
} //End method getPersonnelqtyauth
```

```
/**  
 * method to set the value  
 * @param String  
 * @return void  
 */  
public void setPersonnelqtyreqd(String personnelqtyreqd)  
{  
    this.personnelqtyreqd = personnelqtyreqd;  
} //End method setPersonnelqtyreqd
```

```
/**  
 * method to get the value  
 * @return String  
 *  
 */  
public String getPersonnelqtyreqd()  
{  
    return this.personnelqtyreqd;  
} //End method getPersonnelqtyreqd
```

```
/**  
 * method to set the value  
 * @param String  
 * @return void  
 */  
public void setOk(String ok)
```

```
{
    this.ok = ok;
} //End method setOk

/**
 * method to get the value
 * @return String
 *
 */
public String getOk()
{
    return this.ok;
} //End method getOk

/**
 * method to set the value
 * @param String
 * @return void
 */
public void setPersonneldesc(String personneldesc)
{
    this.personneldesc = personneldesc;
} //End method setPersonneldesc

/**
 * method to get the value
 * @return String
 *
 */
public String getPersonneldesc()
{
    return this.personneldesc;
} //End method getPersonneldesc

/**
 * method to set the value
 * @param String
 * @return void
 */
public void setUnitid(String unitid)
{
    this.unitid = unitid;
```

```
} //End method setUnitid

/**
 * method to get the value
 * @return String
 *
 */
public String getUnitid()
{
    return this.unitid;
} //End method getUnitid

/**
 * method to set the value
 * @param String
 * @return void
 */
public void setPic(String pic)
{
    this.pic = pic;
} //End method setPic

/**
 * method to get the value
 * @return String
 *
 */
public String getPic()
{
    return this.pic;
} //End method getPic

/**
 * method to set the value
 * @param String
 * @return void
 */
public void setPkey(String pkey)
{
    this.pkey = pkey;
} //End method setPkey
```

```
    /**
     * method to get the value
     * @return String
     *
     */
    public String getPkey()
    {
        return this.pkey;
    } //End method getPkey

} //End class

//LoginAction.java

package uobwt;

import java.io.IOException;
import javax.servlet.*;
import javax.servlet.http.*;
import javax.naming.*;
import java.util.Hashtable;
import javax.rmi.PortableRemoteObject;
import javax.ejb.*;
import java.rmi.RMISecurityManager;

import org.apache.struts.action.Action;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;

import uob.*;

/**
 * @author jasmine
 *
```

```
* This servlet class processes user input for log in
*/
public class LoginAction extends Action
{

    public ActionForward execute(
        ActionMapping mapping,
        ActionForm form,
        HttpServletRequest request,
        HttpServletResponse response)
        throws Exception
    {
        // Setup the associated form
        LoginForm loginForm = (LoginForm) form;
        // Get the session variable for the application tier session bean
        HttpSession session = request.getSession();
        UobEJB uobEditor = (UobEJB)session.getAttribute("uobEditor");

        // If the session variable for the application tier session bean is null,
        // initialize a connection to the middle tier
        if (uobEditor == null)
        {
            // if not created yet, create a new one
            uobEditor = this.createReference();

            // Save the bean in session scope
            session.setAttribute("uobEditor",uobEditor);
        }

        // call the application tier session bean to validate user log in
        boolean status = uobEditor.validateUser(loginForm.getName(),
        loginForm.getPassword());

        // reset, so that next login does not see this...
        loginForm.reset(mapping, request);

        //Inform the controlling servlet about success or failure
        if (status != true)
            return (mapping.findForward("error"));
        else
        {
            //request.setAttribute("readerData", readerData);
            return (mapping.findForward("view"));
        }
    }
}
```

```
}

// Initializes a connection to the application tier session bean
// and creates a reference
private UobEJB createReference()
{
    UobEJB uobEditor=null;

    try
    {

        // Get a naming context
        InitialContext jndiContext = new InitialContext();
        //System.out.println("Got context");

        // Get a reference to the UobEJB Bean
        Object ref = jndiContext.lookup("uobejb/UobEJBHome");
        //System.out.println("Got reference");

        // Get a reference from this to the Bean's Home interface
        UobEJBHome home = (UobEJBHome)
            PortableRemoteObject.narrow (ref, UobEJBHome.class);

        // Create a UobEJB object from the Home interface
        uobEditor = home.create();

        return (uobEditor);

    }
    catch(Exception e)
    {
        System.out.println(e.toString());
        return (UobEJB) null;
    }
}
}
```

```

//LoginForm.java
package uobwt;

import javax.servlet.http.HttpServletRequest;

import org.apache.struts.action.ActionErrors;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionMapping;

/**
 * @author jasmine
 *
 * This class defines a form for storage of values entered by a user in the login form
 */

public class LoginForm extends ActionForm
{

    /** name properties */
    private String name = null;

    /** password properties */
    private String password = null;

    /**
     * @see org.apache.struts.action.ActionForm#reset(ActionMapping,
     HttpServletRequest, )
     */
    public void reset(ActionMapping mapping, HttpServletRequest request)
    {
        name = "";
        password = "";
    }

    /**
     * @see org.apache.struts.action.ActionForm#validate(ActionMapping,
     HttpServletRequest, )
     */
    /**
     public ActionErrors validate(
         ActionMapping mapping,
         HttpServletRequest request)
     {
         throw new UnsupportedOperationException("Easy Struts :
         com.acsoft.struts.form.LoginForm.validate(...) not yet implemented.");
     }

```

```
*/  
  
/**  
 * Returns the name.  
 * @return String  
 */  
public String getName()  
{  
    return name;  
}  
  
/**  
 * Sets the name.  
 * @param name The name to set  
 */  
public void setName(String name)  
{  
    this.name = name;  
}  
  
/**  
 * Returns the password.  
 * @return String  
 */  
public String getPassword()  
{  
    return password;  
}  
  
/**  
 * Sets the password.  
 * @param password The password to set  
 */  
public void setPassword(String password)  
{  
    this.password = password;  
}  
}
```

```
//ViewAction.java

package uobwt;

import java.io.IOException;

import javax.servlet.*;
import javax.servlet.http.*;

import org.apache.struts.action.Action;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;

import uob.*;

/**
 * @author jasmine
 *
 * This servlet is called after a user makes a selection in the
 * View window. It returns the identity of the button clicked
 * to the controlling servlet to allow it to process it further
 */
public class ViewAction extends Action
{

    public ActionForward execute(
        ActionMapping mapping,
        ActionForm form,
        HttpServletRequest request,
        HttpServletResponse response)
        throws Exception
    {
        //System.out.println("ViewAction");

        // Return the identity of the clicked button to the controlling servlet
        return mapping.findForward(request.getParameter("action"));
    }
}
```

```

//ViewForm.java

package uobwt;

import javax.servlet.http.HttpServletRequest;

import org.apache.struts.action.ActionErrors;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionMapping;

/**
 * @author jasmine
 *
 * This class defines a form for storage of edit/delete selections
 * made by a user in the View window
 */

public class ViewForm extends ActionForm
{

    /** OptionSelect properties */
    private String OptionSelect = null;

    /**
     * @see org.apache.struts.action.ActionForm#reset(ActionMapping,
     HttpServletRequest, )
     */
    public void reset(ActionMapping mapping, HttpServletRequest request)
    {
        OptionSelect = "";
    }

    /**
     * Returns the OptionSelect.
     * @return String
     */
    public String getOptionSelect()
    {
        return OptionSelect;
    }

    /**
     * Sets the OptionSelect.
     * @param OptionSelect The OptionSelect to set
     */
    public void setOptionSelect(String OptionSelect)

```

```
    {  
      this.OptionSelect = OptionSelect;  
    }  
}
```

## REFERENCES

- [AnAn02] Anderson, G, and Anderson, P. *Enterprise JavaBeans Component Architecture*. Prentice Hall PTR, 2002.
- [BDBS00] Beneventana D., Bergamaschi S., et al. "Information Integration: The MOMIS Project Demonstration." *International Conference on Large Databases, 2000*.
- [FDMS00] "Functional Description of Mission Space (FDMS) Model Representation Data Interchange Format, Version 1.6." Defense Modeling and Simulation Office, November 2000.
- [HaFu01] Haddix, Furman, "Conceptual Modeling Revisited: A Developmental Model Approach for M&S." *Proceedings of 1999 Fall Simulation Interoperability Workshop 2001*.
- [HSH99] Haddix, Furman, Scudder, Roy, Hopkins, Mike, "Unit Order of Battle Toolset." *Proceedings of 1999 Fall Simulation Interoperability Workshop, 1999*.
- [JT02] "J2EE Tutorials" [Java.sun.com/j2ee/tutorial](http://java.sun.com/j2ee/tutorial), 2002.
- [PDS03] Platt, D.S. *Introducing Microsoft .NET*, 3<sup>rd</sup> Edition, Microsoft Press, 2003.
- [SJHF00] Sheehan, Jack, Haddix, Furman, "Knowledge Acquisition for Simulation Requirements: Exchange of Mission Space Models using the Conceptual Models of the Mission Space (CMMS) Data Interchange Format (DIF)." *Proceedings of 2000 Fall Simulation Interoperability Workshop, 2000*.
- [SSJ02] Singh, I., Stearns, B., and Johnson, M. *Designing Enterprise Applications with the J2EE Platform*, 2<sup>nd</sup> Edition, Addison-Wesley, 2002.
- [TPV02] Tulachan, P.V. *Developing EJB 2.0 Components*. Prentice Hall PTR, 2002.
- [UOB99 ] "Unit Order of Battle (UOB) Data Interchange Format (DIF), Version 4.1." Defense Modeling and Simulation Office, March 1999.

[XPS02] XPS notes provided by Jim Lin. His Company had purchased the software from Sequoia Software Corporation, January 2000

[YA96] Yigal Arens et. Al, "Query Reformulation for Dynamic Information Integration." *Journals of Intelligent Information System*, 6(2/3); 99-130, 1996.

## BIBLIOGRAPHY

- Anderson, G., and Anderson, P. *Enterprise JavaBeans Component Architecture*. Prentice Hall PTR, 2002.
- Coulouris, G., Dollimore, J., and Kindberg, T. *Distributed Systems Concepts and Design*, 3<sup>rd</sup> Edition. Addison-Wesley, 2002.
- Farley, J. *Java Distributed Computing*. O'Reilly and Associates, 1998.
- Fisher, M., Ellis, J., and Bruce, J., *JDBC API Tutorial and Reference*, 3<sup>rd</sup> Edition. Addison-Wesley, 2003.
- Habraken, J.W., *Absolute Beginner's Guide to Networking*, 3<sup>rd</sup> Edition. Que Publishing Company, 2001.
- Hall, M. Core, *Servlets and Java Server Pages*. Prentice Hall PTR, 2001.
- Haywood, D., Bond, M., Law, D., Longshaw, A., and Roxburgh, P. *SAMS Teach Yourself J2EE in 21 Days*. SAMS Publishing, 2002.
- [Java.sun.com/j2ee/tutorial](http://Java.sun.com/j2ee/tutorial). (Internet), 2002.
- Lee, R., and Seligman, S. *JNDI API Tutorial and Reference: Building Directory-Enabled Java™ Applications*. Addison-Wesley, 2000.
- O'Neil, J., *Teach Yourself Java*. McGraw-Hill Osborne Media, 1998.
- Platt, D.S., *Introducing Microsoft .NET*, 3<sup>rd</sup> Edition. Microsoft Press, 2003.
- Rolland, F.D., *The Essence of Database*, Prentice hall PTR, 1998.
- Sharma, R., Stearns, B., and Ng, T., *J2EE Connector Architecture and Enterprise Application Integration*. Addison-Wesley, 2001.
- Singh, I., Stearns, B., and Johnson, M., *Designing Enterprise Applications with the J2EE Platform*, 2<sup>nd</sup> Edition. Addison-Wesley, 2002.
- Taylor, A., and Preis, S. *J2EE and Beyond*. Prentice Hall PTR, 2002.
- Tulachan, P.V. *Developing EJB 2.0 Components*. Prentice Hall PTR, 2002.

## VITA

Jasmine Pabby was born in Pune, India, on December 1, 1962. She is the daughter of Iqbal Singh and Harmohan Kaur. She completed her high school at Divine Child, her undergraduate education in accountancy at Mithibai College, and masters in Social Work at Nirmala Niketan, all colleges in Bombay, India. She worked for approximately eleven years as a part-time social worker.

After her separation she came to the United States in Fall, 1998 with her eight-year old son and joined the graduate program in the Computer Science Department at Texas State University-San Marcos (formerly Southwest Texas State University).

Permanent address:           3050 Tamarron Blvd, #3307  
  Austin, TX 78746

This theses was typed by Jasmine Pabby.