COMP 6231 – Assignment 3

Distributed Library Management System using Web Services

By Yogesh Kumar

Winter, 2019

Table of Contents

Introduction

This is a modified/updated version of DLMS in assignment 2. Just like that one, we need to design and develop a distributed library management system consist of different libraries distributed on a local network. A distributed system is a system which consists of multiple components distributed over the network but work together as a single unit of software.

Just like Assignment2, we have three libraries in this assignment as well, namely, Concordia Library, McGill Library and Montreal Library. Each one be replica of each other in sense of functionality but have their own individual databases and information regarding books and library users and other things. We have all the functionality which part of Assignment 2.

We will develop this system using Java web services based on SOAP. With the help of Java web services, an object running in one Java Virtual Machine can invoke methods on another object running in another Java Virtual Machine, just like CORBA but the major difference is that in CORBA, we actually deal with objects and in a way CORBA is more of an object oriented technique where as in web services we deal with message passing in form of XMLs (in SOAP). Both the applications have to be implemented in Java and we can have heterogeneous implementation of communicating ends. We are running different components such as three library instances, user client and manager client on different Java Virtual Machines and with the help of Java JAXWS technology, they will be communicating with each other. So even being distributed into multiple components, the whole system act as a single unit and work as one system. Again for clients, we have used JavaFx technology to build the GUI for user/manager interaction.

As shown in *Figure1.*, the whole application can be divided into components which are described briefly below:

Server Interfaces

This component defines the LibraryOperations interface with JAXWS annotations with couple of other things like PublishURLEnum, UDPPortEnum, OperationEnum and model classes like Book and General exception. As these enums, interface and model classes are used over all the other components, So, it's a good idea to keep them in a separate package and add them in classpath of other components.

Clients

These components are almost similar as in Assignment 2 apart from the fact that now they are working using Web services and are developed using JavaFx. These are GUI components which let users/managers interact with the different libraries and perform different operations.

User client is used by library users to perform operations such as borrow books, return books and find items where as Manager client is used by the library managers to add new items, delete items or list available items. These clients get library servers details from the central repository stub and then further using ORB naming service fetches library stub and invoke methods on those stubs so that appropriate action can take place.

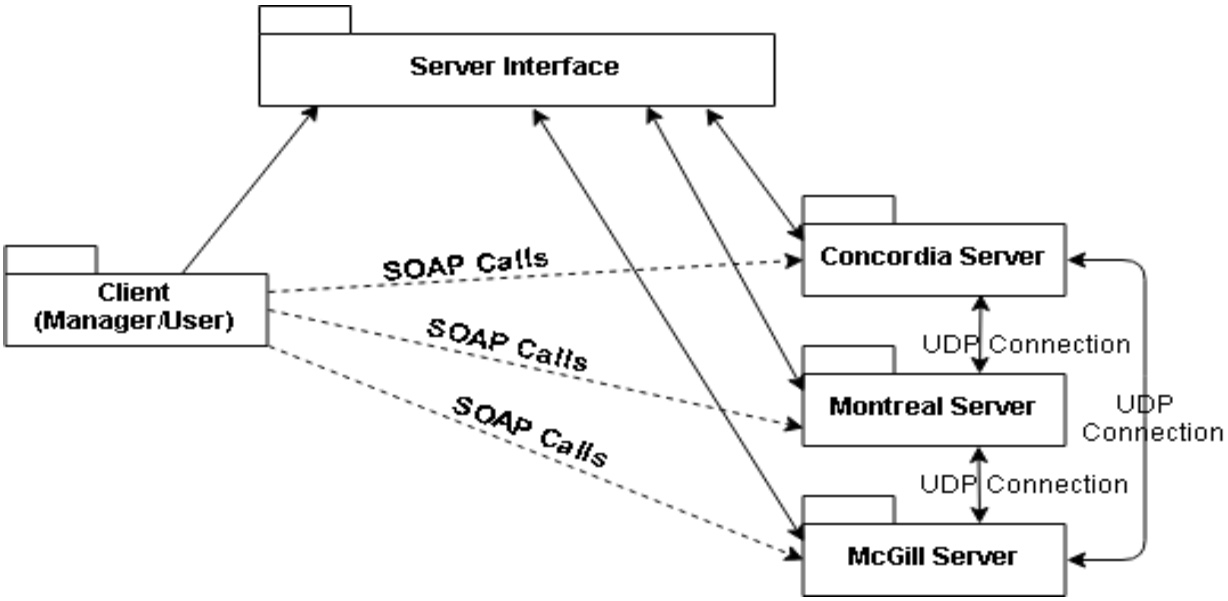

Figure 1. Over all component view of whole system.

Concordia Library/ McGill Library/ Montreal Library

They are similar in functionality as in Assignment 2 but now uses Web services based on SOAP architecture just like everything else. These components represent different libraries, running independently from each other. They have their own database holding information regarding registered users/ managers, books in library, waiting list for books, borrowed items and so on. They also communicate with each other using UDP connection when a user from one library wants to perform some operation on another library.

Whenever a server starts, it fetches details, from ServerInterface component, like url on which this server web service should publish itself and UDP socket connection details etc., The same component also acts as source of repository for clients as well to get those details. And also for different libraries to get the UDP socket details to communicate with other libraries over UDP socket connection.

There is two type of communication here. One is at application layer, through Web service technology, which happen between clients and libraries. And second type of communication is UDP socket connection based communication, which is at transport layer, between libraries.

System Design

Architecture

- Web Service Architecture:

SOAP architecture basically shows how SOAP Web Service act as the mediator between client and server components and help them to communicate. Due to this indirection and agreement on use of an external message passing format (XML in SOAP case) we get remote method invocation ability between heterogeneous applications.

As shown in *Figure2,* there are files in ServerInterface package which are used by JAX-WS to generate the java files using wsgen and wsimport. These java files include mediator which are used to communicated between client and remote server components and interfaces which are implemented by remote components.

All the remote components (Concordia Server, McGill server and Montreal Server) expose an endpoint for the client component to communicate with them and implement an interface which let client know what all operations can be performed by these services. Client application interact with stubs generated by JAX-WS to interact with these exposed endpoints and execute the methods on server components remotely.

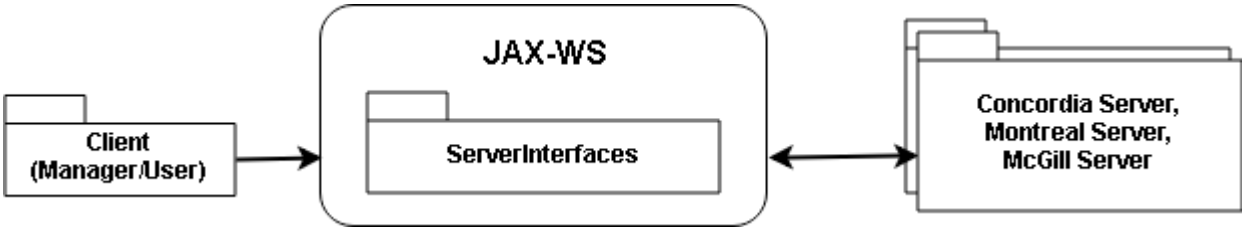

Figure 2. CORBA Architecture Diagram

• Application Architecture:

The application architecture is same as in Assignment 2 but using SOAP based web service for implementation instead of CORBA and is explained below:

The architecture of this distributed application is MVP(model-view-presenter) which is a which is a variation of MVC(model-view-controller) as, unlike MVC, in this architecture model is not responsible for updating the view which is JavaFx based rather controller is responsible to do so once it receives the result from the remote methods which are exposed by objects running on library servers, it invokes through CORBA stubs.

As shown in *Figure3.,* we have two clients, manager client and user client, which have JavaFx based view which is backed by respective controllers. Users/managers can interact with whole distributed application using these clients and GUI based view. Through these views, event is triggered by users which are captured by the controller classes. Based on the type of event triggered and parameters passed by GUI user, appropriate web service server details are fetched from the server interface component which for these sort of requests act as model. Once controller have details of library server's endpoint, it invokes the methods on the library server classes using middleware and JAX-WS. These model class then make local calls to the in memory database class to do the computation.

Class diagram

The most complicated component, in terms of functionality and implementation, of our distributed application are three libraries which are replicas of each other in terms of functionality they provide and business logic implementation. But the only difference is that they are configured and uses different ports

for UDP connection and have different individual databases. So the dependencies and functionalities of various classes are shown in class diagram, in *Figure4*, which represent Concordia Library.

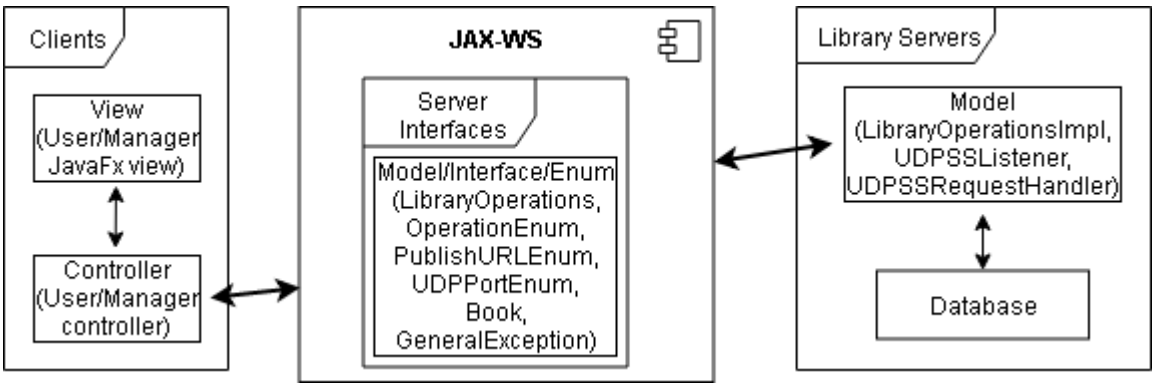

Figure 3. Application Architecture Diagram

Class Description:

ConLibraryServerMain: Just like in Assignment 2, this is the entry point of library server component and there is not major change functionality wise apart from the fact that now it used JAX-WS classes to to host the library server as web service.

LibraryOperations: This is an interace class for library operations, which is mention all the library servers in LibraryOperationsImpl java class to provide the business logic to methods in this interface. So that those methods can be called by client classes using library operations stubs fetched from JAX-WS.

LibraryOperationImpl: This class is the extends of LibraryOperations interface and provides business logic to those methods. This class provide total 10 operations out of which 7 are as per assignment requirements which are Add Item, Remove Item, List Item, Borrow Item, Find Item, Return Item, exchange Item. Additional 3 operations are to check if given user exists or not and to add a user to waiting list and its overloaded version.

Book: This class represents an entity class for book.

GeneralException: This is a custom exception class. This exception is used to handle application's operation based custom exceptions.

PublishURLEnum: This enum is used to define URLs for web service publishing as constants so they can be used by library servers as well as by clients.

UDPPortEnum: This enum have port numbers as constants for UDP connection for all three libraries.

OperationsEnum: This enum define the operation as constants which are invoked by one library server to another over UDP connection.

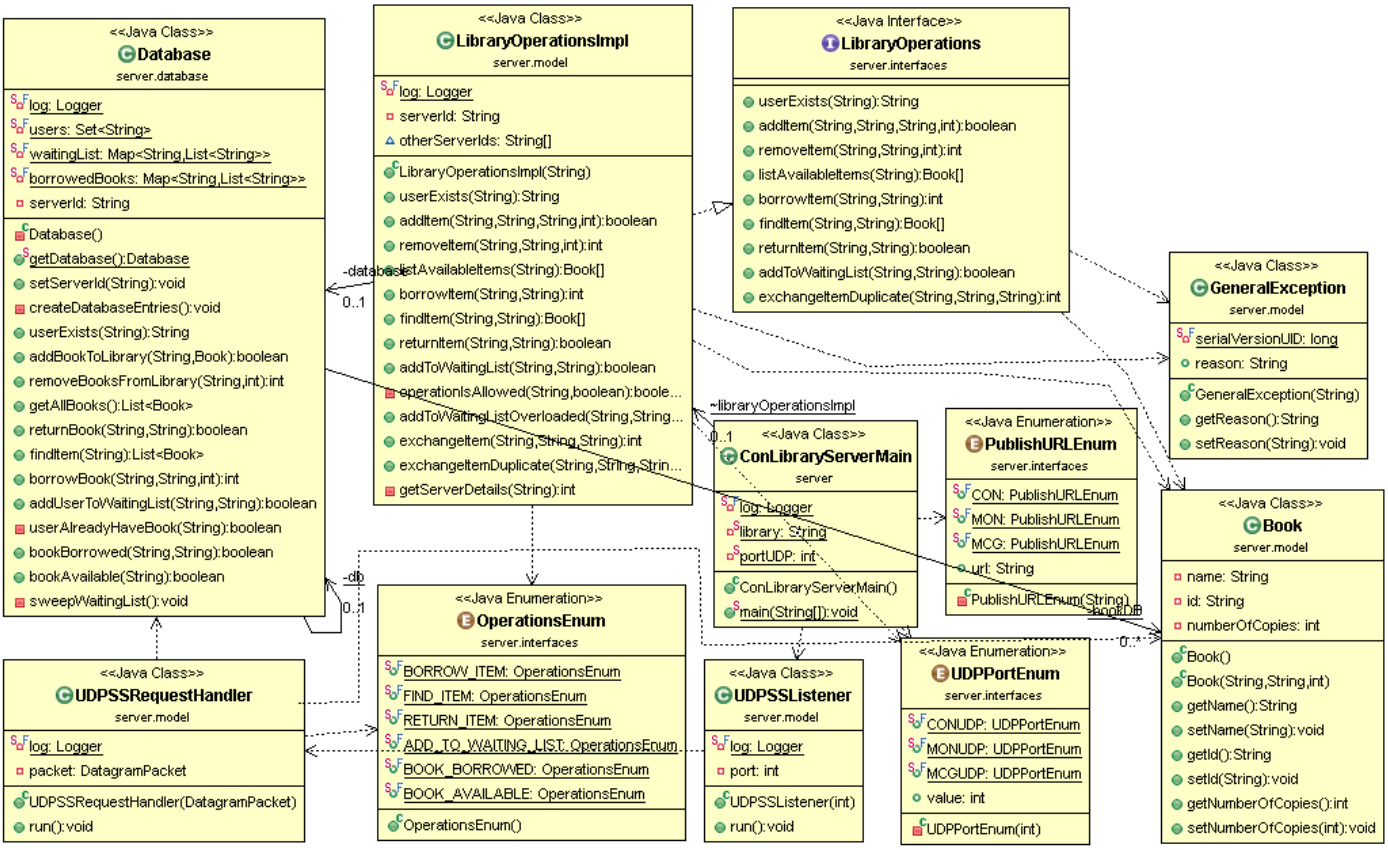

Figure 4. Class Diagram of library component.

Database: This class holds the application data in various data structures. This act is the in memory database for library. It has a **users: Set<String>** which act as the ids of users registered with this library, it also has a **bookBD: Map<String, Book>** which is used to store the details of books which library have. Each book instance is mapped against a bookId in this map. It has a **waitingList: Map<String,List<String>>** which is used to store all the user waiting for a book in a list of string mapped to that book id. Another data structure is **borrowedBooks: Map<String, List<String>>** which holds a list of users, who borrowed a particular book from library, mapped against the borrowed book id.

This class also provides with various methods to manipulate these data structures in synchronized thread-safe way to support the functionalities offered by library operations.

UDPListener: This class listens to a specific port to handle all the UDP requests made on that port by other libraries. Once it gets a request, it created another thread to handle that request and pass the details of the request to that thread to do processing and return the result back to requesting library.

UDPRequestHandler: This is the class which handle the request made by other libraries and captured by UDPListener thread. Based on the details which are passed by UDPListener to this class, this class do the required processing and handle the request by invoking required methods on database and sending the requesting library response of these database invocation calls.

Use Cases

As all the functionality of Assignment2 as similar to Assignment 1 so we will be not doing rigorous testing of previous functionality but just testing few cases but for new exchange item functionality we will be checking for various cases.

For testing purpose each server have few users'/managers registers to it and a list of books. List of users registered to each library (with starting 3 character of username being different which is according to library server) is CONM1111, CONM1112, CONM1113, CONM1114, CONM1115, CONU1111, CONU1112, CONU1113, CONU1114, CONU1115, CONU1116, CONU1117, CONU1118, CONU1119, CONU1120.

Similarly books in library database (starting three character of book id is as per the library server id) is as follow CON6231, CON6641, CON6491, CON6651, CON6481 with their corresponding book names as Distributed Systems, Advanced Programming, Systems Software,Algorithm Design, System Requirements Spec.

Method to test: **Borrow Book()**

In this test case we want to test if multiple books can be borrowed by a library user from his own library or not. We make request two times and it should be successful both times.

userID	ItemId	ItemName	Result
CONU1112	CON6641		Book is issued to CONU1111
MONU1112	CON6641		Book is issued to MONU1111

Method to test: **Find Item()**

Given an item name we want to test if user client can find matching books from all libraries or not. Client should show item ids and quantity of books available on all the servers whose names match with the provided name.

userID	ItemId	ItemName	Result
CONU1111		Distributed Systems	CON6231 3, MCG6231 5, MON6231 4

Method to test: **Return Item()**

Case1: User CONU1111 have 1 MON6231 book. We will try to return MON6231 2 times. User should only be able to return book once as he has only 1 MON6231 issues on his name.

userID	ItemId	ItemName	Result
CONU1111	MON6231		Book returned to library successfully.
CONU1111	MON6231		Unable to return book to library

Method to test: **List Item()**

We will provide manager id to manager client and query for listing the items available at the serer. It should return all the items available in the library.

userID	ItemId	ItemName	quantity	Result
CONM1111				CON6231 Distributed Systems 5,CON6641 Advanced Programming 5, CON6491 Systems Software 5,CON6651 Algorithm Design 5, CON6481 System Requirements Spec 5

Method to test: **Add Item()**

We will add an item with correct item id and the server should add the item to its database.

userID	ItemId	ItemName	quantity	Result
MCGM1111	MCG1234	Temp book	2	Book is added to library.

Method to test: **Remove Item()**

First we will try to delete newly added item more than the quantity in which its available and server should not delete anything but just let user know that it can't perform operation and lastly we will try to delete all the copies of this item and server should be able to perform this operation and inform the manager through client.

userID	ItemId	ItemName	quantity	Result
MCGM1111	MCG1234	Temp book	3	Can't delete more books than library currently have.
MCGM1111	MCG1234	Temp book	-1	Item is completely deleted from library.

Method to test: **Exchange Item()**

Case1: First with user CONU1111 we borrow book CON6231 and then exchange it with MON6231.

Case2: Then we again try to borrow book CON6231 and exchange it with another book from same library say MON6641, in this case as user already have a book, MON6231, library should let us exchange the book.

userID	NewItemId	OldItemId	Result
CONU1111	MON6231	CON6231	Book exchange is successful.CONU1111
CONU1111	MON6641	CON6231	Unable to exchange book.