Introduction

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.

In this, distributed library management system, there are three libraries namely as Concordia Library, McGill Library and Montreal Library. These libraries instances are on same network but can be distributed among different computers. There will be users and managers interacting with these libraries regarding different tasks based on their roles and operations exposed by libraries. Each library has its own database which stores information regarding available books, library users and mangers, borrowed books detail, details of users waiting for books to be issued to them. There are different operation belonging to mainly two different roles, that is, manager and users.

We will develop this system using Java RMI. With the help of Java RMI, an object running in one Java Virtual Machine can invoke methods on another object running in another Java Virtual Machine. These programs can also be running on same or different machines but they should run on same network to make this method invocation on other Java Virtual Machine's objects possible. In our case we are running different components such as three library instances, user client, manager client and central repository on different Java Virtual Machines and with the help of Java RMI 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. 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 5 components which are described briefly below:

Central Repository

This component act as a repository for other components. It holds the information for the library servers regarding the RMI port, stub details and also the UDP socket connection details with which different library servers communicate with each other. Whenever a library server starts, it registers its details with the central repository so that the clients can get the required details from this repository to connect and communicate with required library server.

Clients

These components 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 and then using Java RMI, invoke methods on those libraries so that appropriate action can take place.

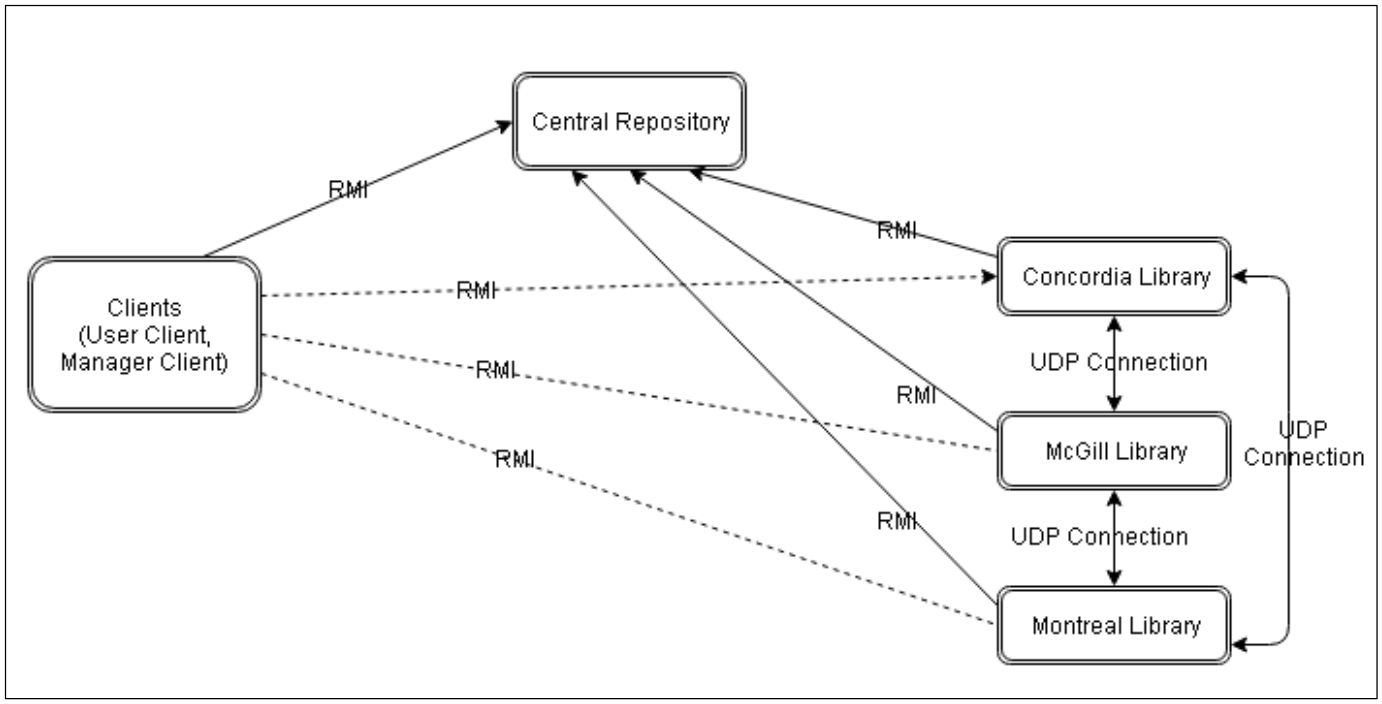

Figure 1. Over all component view of whole system.

Concordia Library/ McGill Library/ Montreal Library

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, its details, like port number on which stub is registered and stub name and UDP socket connection details etc., are registered with the central repository for the clients to use that information and communicate with the libraries. And also for different libraries to get the UDP socket details from central repository to communicate with other libraries over UDP socket connection.

There is two type of communication here. One is at application layer, through RMI technology, which happen between clients, central repository and libraries. And

second type of communication is UDP socket connection based communication, which is at transport layer, between libraries.

System Design

Architecture diagram

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 RMI.

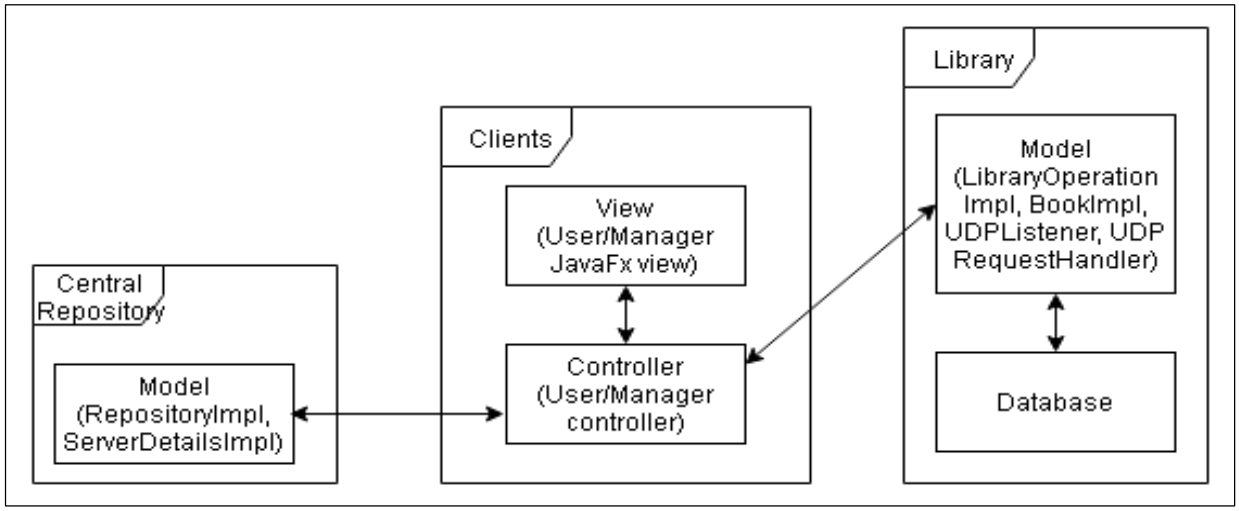

Figure 2. Architecture Diagram

As shown in *Figure2.,* 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 server details are fetched from the central repository which for these sort of requests act as model. Once controller have details of library server's, it invokes the methods on the library server model classes. 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 exposed on different ports and have different individual databases. So the dependencies and functionalities of various classes are shown in class diagram, in *Figure2.*, which represent Concordia Library.

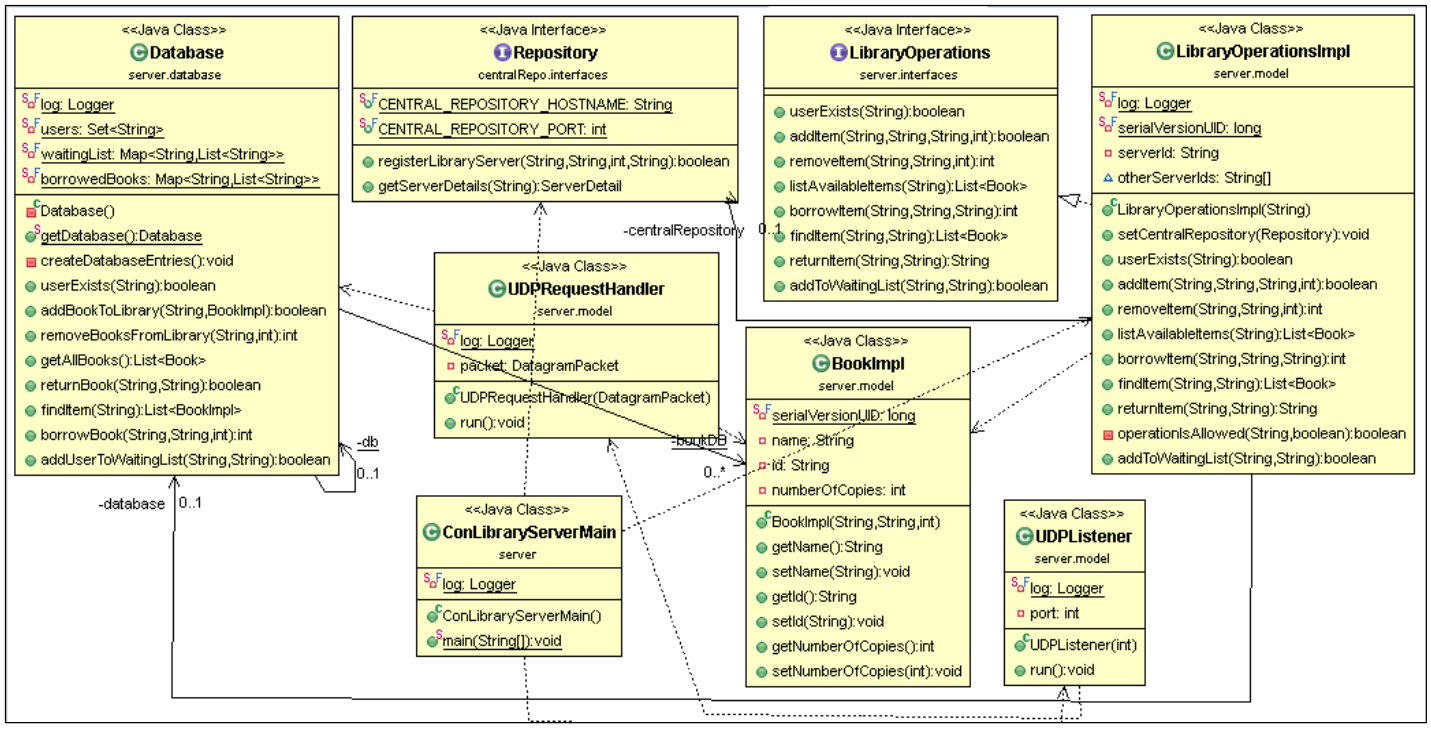

Figure 3. Class Diagram of library component.

Class Description:

ConLibraryServerMain: This is the entry point of library server component. This class will setup the whole component and register the stud to the registry at a specific port so that clients can access it without any problem. It also opens a thread based UDP connection socket to listen to other library's calls and also register those details with central repository.

Repository(Interface): This is an RMI interface used by all the servers and clients to communicate with central repository to either register server details or fetch those details.

LibraryOperations(Interface): This is another RMI interface which is implemented by all the library servers and used by clients to interact with the library servers to perform

different operations.

BookImpl: This class is the implementation of Book interface. This is a POJO class which represent a book in application.

LibraryOperationImpl: This class is the implementation of LibraryOperation interface which provide operations which need to be performed with in a library based on the assignment. This class provide total 8 operations out of which 6 are as per assignment requirements which are Add Item, Remove Item, List Item, Borrow Item, Find Item and Return Item. Additional 2 operations are to check if given user exists or not and to add a user to waiting list.

This class is also responsible to check if the user/manager request can be fulfilled by this library instance or this library needs to communicate with other libraries to get the request fulfilled.

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.

Sequence diagram

There are two types of operations based on the component communications which are as follows:

1. Requests from client is handled by the user's /manager's library component without the need of any additional communication with other libraries to fulfill that request. All manager request comes under this type.
2. Requests where user's library needs to communicate with other libraries to fulfill user's request. All user's request can be put in this category.

Sequence diagram in *Figure 3* shows flow for both type of requests followed by explanation.

addItem(): This is a manager level operation. Manager request for this operation using manager's JavaFx GUI client and provided required detail to the client. When manager submit the request the controller of JavaFx GUI captures the request and based on passed request parameters fetch the required library server details from central repository components through RMI. Once controller have the server details then it invokes the remote method on LibraryOperation class in Library component to add the item to that library database. LibraryOperation methods will further invoke the local method on Database class to add the item to the library and return.

findItem(): This is a user level operation. Library user make a request through the user client's JavaFx GUI to get the details of all the items available with different libraries with a specified name. Once the request

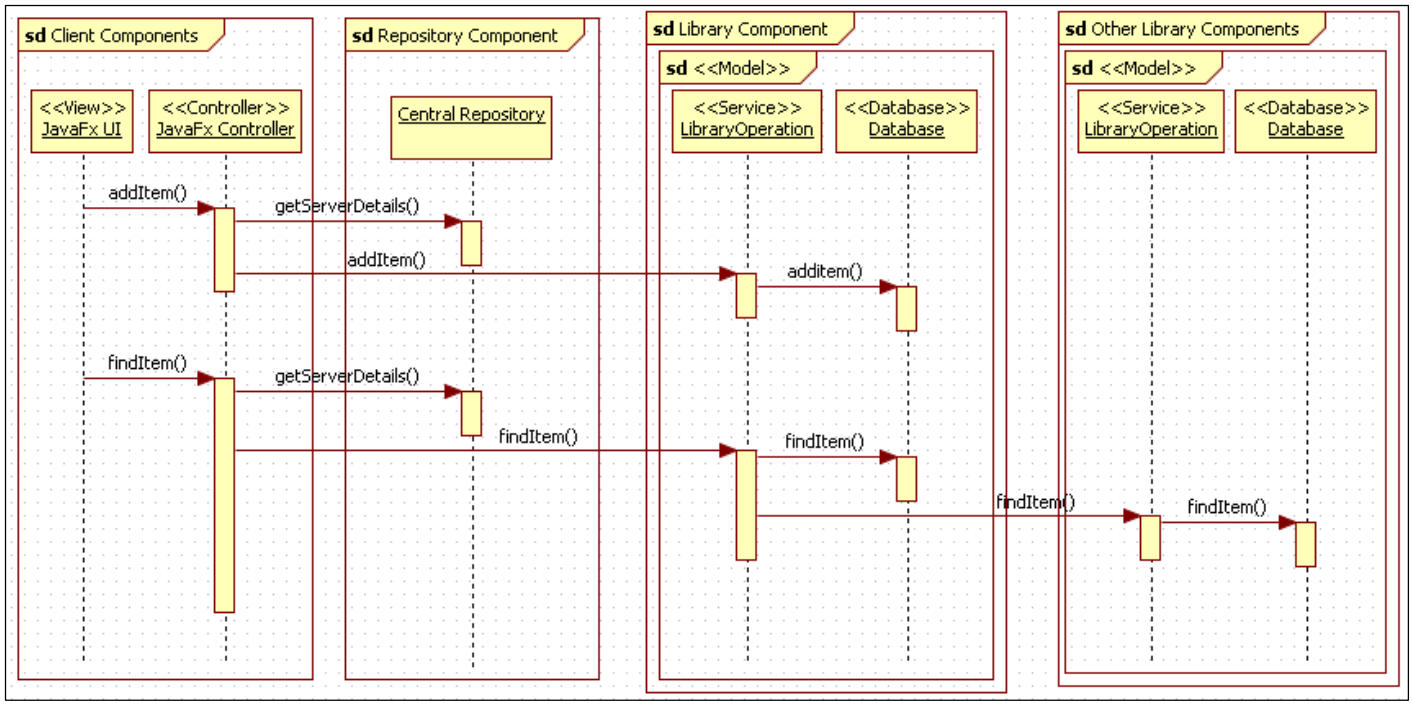

Figure 4. Sequence Diagram

is capture by controller class in client component, the controller will fetch the server details to which this user belongs through RMI. Using these server details, client's controller object will remotely invoke findItem() method on its library server. As per the requirements, user should get books detail from all the libraries which have the name for which user have made the request, so now user's library will communicate, over UDP socket connection, with other library servers as well to get the details of book they have with user specified name. Once user's library gets the response from other libraries as well then it will gather the data and send it back to user.

Use Cases

We will be testing user client first and then followed by manager client. 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, CON6501, CON6411,CON6180, CON6461, CON6521 with their corresponding book names as Distributed Systems, Advanced Programming, Systems Software,Algorithm Design, System Requirements Spec, Programming Competency, Comparative Studies,Data Mining, Software Design, Advance Database.

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

Case1: 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
CONU1111	CON6231		Book is issued to CONU1111
CONU1111	CON6231		Book is issued to CONU1111

Case2: In this test case we want to test if multiple books can be borrowed by a library user from other university library or not. We make request for same book twice and its should not issue book second time.

userID	ItemId	ItemName	Result
CONU1111	MON6231		Book is issued to CONU1111
CONU1111	MON6231		This user already have one copy of this book from this library.

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

Case1: 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 2 CON6231 and 1 MON6231 book. We will try to return CON6231 3 times and MON6231 2 times. In first case return book should be successful in first 2 times but user should not be able to return third time and as for MON6231, user should only be able to return book once as he has only 1 MON6231 issues on his name.

userID	ItemId	ItemName	Result
CONU1111	CON6231		Book returned to library successfully.
CONU1111	CON6231		Book returned to library successfully.
CONU1111	CON6231		Unable to return book to library
CONU1111	MON6231		Book returned to library successfully.
CONU1111	MON6231		Unable to return book to library

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

Case1: 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. And also we will try to do this operation on manager

Client with user id and the server should not let us perform this operation.

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, CON6501 Programming Competency 5, CON6411 Comparative Studies 5, CON6180 Data Mining 5, CON6461 Software Design 5,CON6521 Advance Database 5
CONU1111				RemoteException occurred in server thread; nested exception is: java.rmi.AccessException: Operation is not allowed for this USER.

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

Case1: First we will try to add an item whose item id starts with different server id than the one with which manager is register and in this case manager client should not let us proceed. Second with will add an item with correct item id and the server should add the item to its database.

userID	ItemId	ItemName	quantity	Result
MCGM1111	CON1234	Temp book	2	Item id should start with MCG
MCGM1111	MCG1234	Temp book	2	Book is added to library.

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

Case1: 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. Secondly we will delete single copy of this item and server should be able to decrement the number of copies of this item 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	1 books related to MCG1234 are removed from library.
MCGM1111	MCG1234	Temp book	-1	Item is completely deleted from library.