DESIGN DOCUMENTATION

The following documentation comprises of 6 sections giving information about various techniques, architecture, data-structures and challenges faced for the development of Distributed Library Management System.

Task at Hand

The assignment comprised of design and implementation of Distributed Library Management System for a group of libraries: a distributed system used by library managers to manage the information about the items available in the libraries and library users to borrow or return items across the libraries.

We considered three libraries namely Concordia (CON), McGill (MCG) and Montreal (MON) for the implementation. Along with the libraries, there should be manager client to handle all the functionalities available to the manager and user client to perform all the user functionalities.

There is also log files for every user, manager and library which maintains history of all the operations that have been performed.

Also, a constraint that the design of the server should maximize concurrency. In other words, using proper synchronization that allows multiple users to perform operations for the same or different records at the same time

The documentation comprises of:

TECHNIQUES

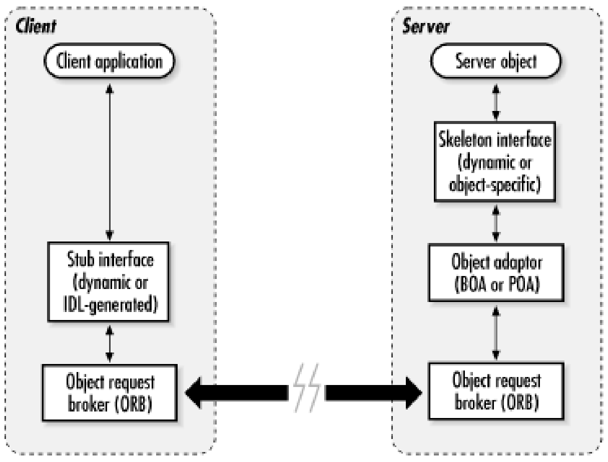
The Distributed Library Management System is developed using 2 useful techniques at its core.

1) CORBA
2) UDP Socket Programming

The Common Object Request Broker Architecture (CORBA) is a standard developed by the Object Management Group (OMG) to provide interoperability among distributed objects. CORBA is the world's leading middleware solution enabling the exchange of information, independent of hardware platforms, programming languages, and operating systems. CORBA is essentially a design specification for an Object Request Broker (ORB), where an ORB provides the mechanism required for distributed objects to communicate with one another, whether locally or on remote devices, written in different languages, or at different locations on a network.

The CORBA Interface Definition Language, or IDL, allows the development of language and location-independent interfaces to distributed objects. Using CORBA, application components can communicate with one another no matter where they are located, or who has designed them. CORBA provides the location transparency to be able to execute these applications.

UDP stands for User Datagram Protocol. UDP Socket Programming is used for server to server communication. UDP is connectionless protocol and is faster than TCP. The system comprises operations involving forwarding request to other library for borrowing a book, finding a book, etc. All these operations are carried out using the UDP Socket programming. A server is always in listening mode, and performs request from other server and sending appropriate response according to the constraints.


Client
Client application
Stub interface (dynamic or IDL-generated)
Object request broker (ORB)
Server
Server object
Skeleton interface (dynamic or object-specific)
Object adaptor (BOA or POA)
Object request broker (ORB)

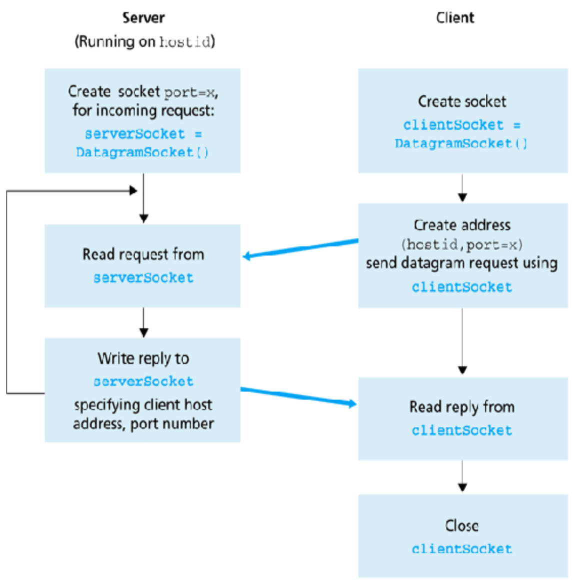

Server (Running on hostid)
Client
Create socket port=x, for incoming request: serverSocket = DatagramSocket()
Create socket clientSocket = DatagramSocket()
Read request from serverSocket
Create address (hostid, port=x) send datagram request using clientSocket
Write reply to serverSocket specifying client host address, port number
Read reply from clientSocket
Close clientSocket

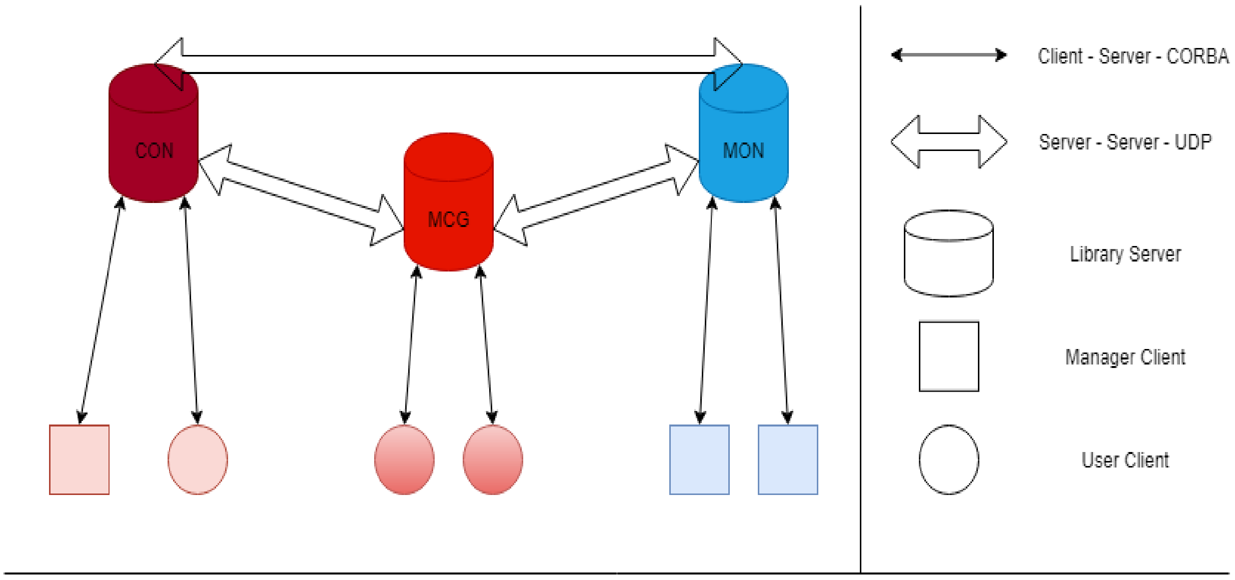
Fig 1.1 CORBA Architecture

Fig 1.2 UDP Socket Client-Server

ARCHITECTURE

The architecture of the system involves 3 library servers bound at a specific port number with a specific name. This port number is used by clients and other servers to connect and perform operations. Concordia (CON) at port 3333, Montreal (MON) at port 5555 and McGill (MCG) at port 4444.

There can be multiple clients running parallel to each other. Clients are of 2 types namely Managers and Users and have different set of functionalities under their domain. The following architecture allows us to have following core functionalities of a distributed system namely concurrency, scalability, and openness.



DATA STRUCTURES

Data is the key to any system. A data structure is a data organization, management and storage format that enables efficient access and modification. More precisely, a data structure is a collection of data values, the relationships among them, and the functions or operations that can be applied to the data.

In this system we require to store three important type of data associated with server and clients and they are

1. Library inventory which consists information of all the books
2. Borrowers list which maintains a list of all the user having books from the library
3. Waitlist queue which helps maintains waitlist for the books

To satisfy the requirements, this system uses a complex data structure of HashMap. HashMap is a Map based collection class that is used for storing Key & value pairs, it is denoted as HashMap<Key, Value> or HashMap<K, V>. This class makes no guarantees as to the order of the map. It is similar to the Hashtable class except that it is unsynchronized and permits nulls(null values and null key).

HashMap works on the principle of hashing, we have put() and get() method for storing and retrieving object from HashMap. When we pass both key and value to put() method to store on HashMap, it uses key object hashcode() method to calculate hashcode and them by applying hashing on that hashcode it identifies bucket.

Following HashMaps are used to successfully store and use important data.

1. HashMap<String,Book> library
2. HashMap<String,ArrayList<Book>> borrowers
3. HashMap<String,ArrayList<String>> waitlist

Book is a model element which stores Book ID, Name, and Quantity and allows to update it using getter setter methods.

TESTED SCENARIOS

Test Scenario #1

Library contains item CON1111 – JAVA – 10.

Manager adds item CON1111 – C++ - 10.

Result: FAIL

Reason: If itemID already exists in Library, when added again both itemID and itemName should match the already added item.

Test Scenario #2

Library contains item CON1111 – JAVA – 10.

Manager removes item CON1111 – 100.

Result: FAIL

Reason: If entered quantity is greater than the current quantity in library, it should not do anything and throw error message to user.

Test Scenario #3

MCGUser has item CON1111.

MCGUser tries to borrow item CON2222.

Result: FAIL

Reason: Should not allow user to get the book as the user already has 1 book issued and must also be not allowed to get on waitlist for this item.

Test Scenario #4

MCGUser has item CON1111.

MCGUser tries to borrow item MON2222.

Result: SUCCESS

Reason: Should allow user to get the book from other server as long as it hasn't reached limit of 1 book per other server.

Test Scenario #5

Library contains item CON1111 – JAVA – 0. And has 3 users in waitlist.

Manager adds 2 copies of CON1111. First 2 users in waitlist get the books.

Result: SUCCESS

Reason: Whenever the item is incremented in library waitlist needs to be checked in order and books should be issued accordingly.

Test Scenario #6

MCG USER with MCG1234

Exchange MCG0001 MCG9999

Result: FAIL

Reason: MCG USER doesn’t have MCG0001.

Test Scenario #7

MCG USER with MCG1234

Exchange MCG1234 MCG1234

Result: FAIL

Reason: MCG USER already has MCG1234.

Test Scenario #8

MCG USER with MCG1234, MCG0001

Exchange MCG0001 MCG1234

Result: FAIL

Reason: MCG USER already has 1 copy of MCG1234.

Test Scenario #9

MCG USER with MCG1234

Exchange MCG1234 CON1234

Result: SUCCESS

Reason: CON1234 is available and MCG1234 is also borrowed by the user. Satisfies both.

Test Scenario #10

MCG USER with MCG1234, CON1234

Exchange MCG1234 CON9999

Result: FAIL

Reason: MCG USER already has a book from CON. Should not allow to borrow one more.

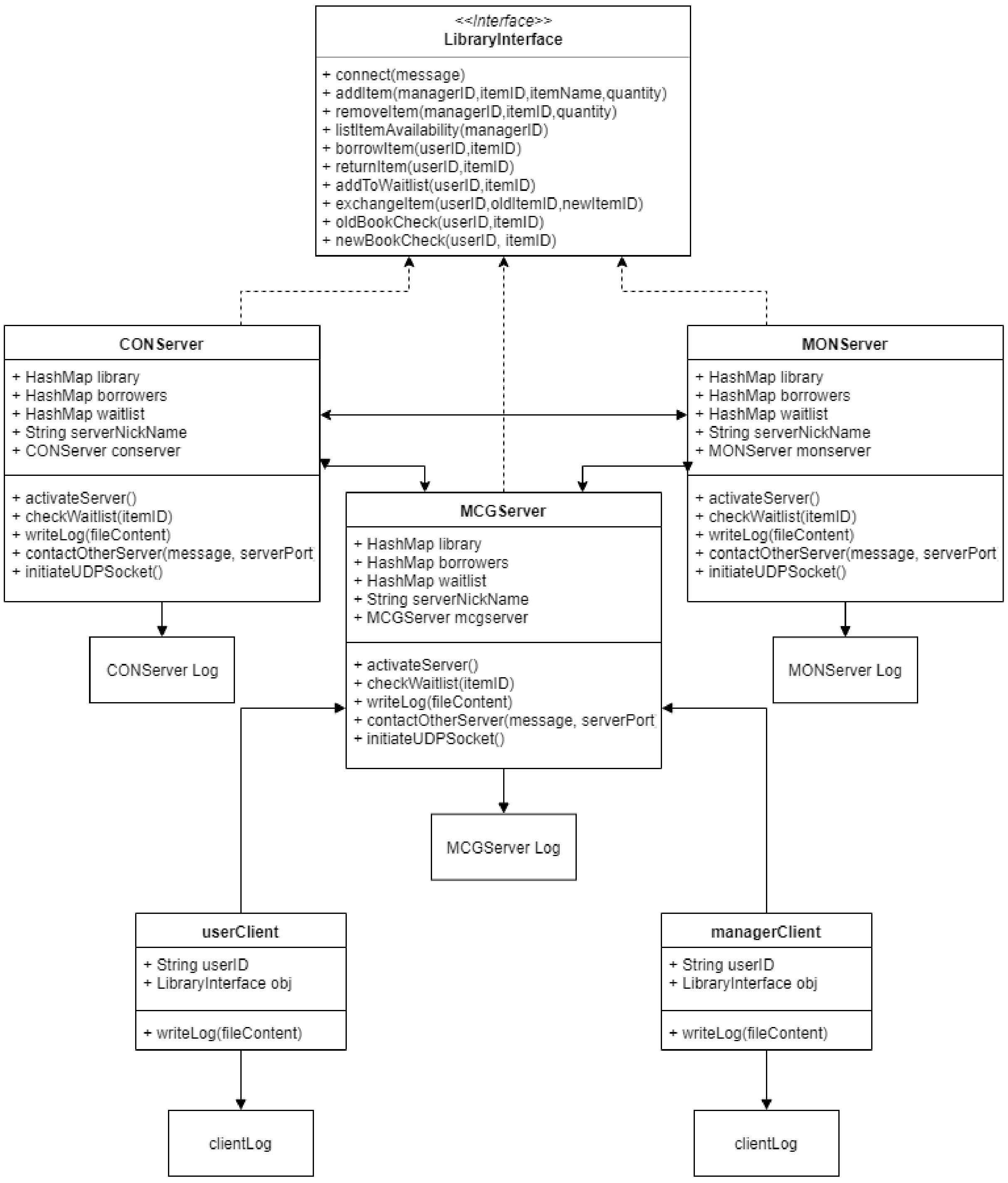
Test Scenario #11

CON USER with MCG1234, CON1234

Exchange MCG1234 MON0001 Result: SUCESS

Reason: CON USER has no book from MON. Also has borrowed MCG1234. Satisfies both.

CLASS DIAGRAM



IDL FILE

```
module repository{
      interface LibraryInterface{
      string connect(in string message);
      string addItem (in string managerID,in string itemID,in string
itemName,in long quantity);
      string removeItem (in string managerID,in string itemID,in long
quantity);
      string listItemAvailability (in string managerID);
      string borrowItem (in string userID,in string itemID);
      string findItem (in string userID,in string itemName);
      string returnItem (in string userID,in string itemID);
      string exchangeItem (in string userID,in string newItemID,in string
oldItemID);
      string oldBookCheck(in string userID,in string itemID);
      string newBookCheck(in string userID,in string itemID);
      string addToWaitlist(in string userID, in string itemID);
      string serverRequestFind(in string userID, in string itemName);
      long checkBorrowedBooks(in string userID);
      };
};
```

CHALLENGES

This is the first time I've developed a distributed application and faced many challenges during the process of its development. Following are few of the challenges faced and the process to solution.

1. Design and Architecture of DLMS
   Solution: Few raw diagrams on paper and a bit of development cleared the doubts regarding proper flow and design overview of the system.

2. Server to Server Communication using UDP
   Solution: Since we cannot directly invoke a method in other server using UDP the way we can using RMI, so String message content sent to server contained important information that tells the server which method to call and sends response accordingly.

3. Concurrency and Data Integrity
   Solution: Multithreading and serializing methods to achieve both concurrency and data integrity.

4. Concurrent Modification
   Solution: Serializing object and passing instance and then updating the data structure so we do not face Concurrent Modification Exception.

5. Efficient Data Access
   Solution: Suppose there are thousands of book in library and hundreds of records in borrowers and waitlist. We need quick access to the specific data that we require in the least amount of time. For that we've used HashMap data structure which allows to retrieve and put data with time complexity of O(1).

6. CORBA Port Issue
   Solution: Was unable to run CORBA code on 1050 port. So changed the ORBD port to 6666 and was able to run successfully