**COMP6231 Assignment 2**

**Distributed Appointment Management System using CORBA**

**Darshak Kachchhi (40206619)**

Instructor: R. Jayakumar

TA: Brijesh Lakkad

**Computer Science & Engineering**

**Concordia University**

**Montreal, Quebec**

**Table of Contents**

1. Overview 3
   1. Tools 3
2. Create & Run System 4
3. Architecture 5
4. Features 6
5. Test Cases 7

Already Added Test Data 7

* 1. Add appointment 7
  2. Book appointment 8
  3. Remove appointment 8
  4. List appointment 9
  5. Get appointment Schedule 9
  6. Cancel appointment 9

1. **Overview**

The Distributed Appointment Management System (DAMS) is a distributed system for health care; It is used by an admin of the hospitals who manages the information about the medical appointments and patients to book or cancel a medical appointment across three different hospitals Montreal (MTL), Sherbrooke (SHE), and Quebec (QUE) within a system.

Hospital admins and patients are uniquely identified by the admin id (e.g., MTLA0000) and patient id (QUEP2981) respectively. There are 3 types of appointment types for which slots can be created by the admin: Physician, Surgeon, Dental. There are three-time slots are available for each appointment type in a day. Each appointment type is a combination of city, appointment slot, and date.

Each server maintains its database using the HashMap. Client and Server are communicating using the CORBA. While inter-server communication is done by the UDP communication. Each server maintains a log file for all the operations performed by the server. Also, for each patient and admin client log file is maintained.

To make the system more robust, inter-server communication is done using the thread. Since there are multiple users are accessing the server concurrently, the proper synchronization of data is implemented in the code. All the user inputs are case insensitive.

* 1. Tools
* Java IDE Eclipse
* Java JDK version 1.8

1. **Create & Run System in Eclipse**

**To create a Distributed system using CORBA there are 5 steps.**

* 1. Create a Remote Interface (HospitalServer.idl) using the OMG's Interface Definition Language (IDL).
  2. Compile the remote interface using the command “**idlj -fall HospitalServer.idl**”. It will generate a skeleton to put together server application.
  3. Implement a Server (MTLHospitalServer.java, SHEHospitalServer.java, QUEHospitalServer.java) by extending HospitalServerInterfacePOA class.
  4. Implement the Client application (AdminClient.java and PatientClient.java).
  5. Start the applications

**Once we have implemented a server and a client, we can start the name service,**

start orbd -ORBInitialPort 900

**Run command of Server files**

javaMontrealServer -ORBInitialPort 900 -ORBInitialHost localhost

java SherbrookeServer -ORBInitialPort 900 -ORBInitialHost localhost

java QuebecServer -ORBInitialPort 900 -ORBInitialHost localhost

**Run command of AdminClient.java file**

java AdminClient <AdminId> -ORBInitialPort 900 -ORBInitialHost localhost

**Run command of PatientClient.java file**

java PatientClient <PatientId> -ORBInitialPort 900 -ORBInitialHost localhost

**Order to run the system**

* Start Montreal, Sherbrooke, and Quebec server in any order
* Start either AdminClient or PatientClient based on requirements.

1. **Architecture**

There are three different servers MTL, QUE, and SHE. When all these servers are started, all the servers start their own UDP servers for communicating with the patient-client and server client. These servers are running all the time to listen to requests from clients.

Depending on the client ID of the patient or admin, the client system will connect you to the respective server by doing look up from the NamingService and It allows client to call a service provided by the server.

A client only communicates with their corresponding server. But there are multiple options (list appointment availability, book appointment, get schedule appointment of a patient, cancel appointment and Swap appointment) which have required to communicate with the other servers. This communication is done by the UDP socket communication. The client-server makes a UDP request to other servers concurrently and it will get the response from the other servers, and it is returned to the client.

1. Features
2. Server Database: each server has its own database, and it is implemented using HashMap.

Graphical user interface

Description automatically generated

1. Appointment Details are maintained by AppointmentDetails class object which has details of type of appointment, appointment id, capacity and list of patient id who has booked the slot.

Text, letter

Description automatically generated

1. Log has been implemented to track all the activities on each server as well as for each admin and patient who use the system.

Format of the log:

Request Date Time | Request Type | Request Parameters | Server Response | Status of Completion

Text

Description automatically generated

1. **Test Case**

**Already Added Test Data**

|  |  |  |  |
| --- | --- | --- | --- |
| **Server Name** | **Appointment Type** | **Appointment ID** | **Patient List** |
| Montreal | Physician | MTLA030222 | **MTLP2345, QUEP5465** |
| MTLE030222 | **MTLP1245, MTLP2463, MTLP9875** |
| **MTLM010222** | **MTLP2345, MTLP9875, MTLP3246** |
| **Dental** | **MTLM030222** | **MTLP2345, MTLP3246** |
| **MTLE030222** | **MTLP2345** |
| **MTLA010222** | **MTLP3246, MTLP1245, MTLP2463, MTLP5465** |
| **MTLA020222** | **MTLP2345, MTLP5465** |
| **Surgeon** | **MTLA030222** | **MTLP1245, MTLP2463** |
| **MTLM030222** | **MTLP3246, MTLP9875** |
| **Quebec** | **Physician** | **QUEA040222** | **MTLP2345, QUEP5465** |
| **Dental** | **QUEA010222** | **QUEP5465** |
| **QUEA020222** | **QUEP5465** |
| **Sherbrooke** | **Physician** | **SHEE080222** | **MTLP2345, SHEP5565, SHEP2475** |
| **Dental** | **SHEA050222** | **SHEP5565, SHEP2475** |
| **Surgeon** | **SHEE070222** | **MTLP2345, SHEP5565, SHEP2475** |

**Test Data**

|  |  |  |
| --- | --- | --- |
| **Test Method** | **Expected Output** | **Actual Output** |
| Add appointment | **Success: Appointment Added** |  |
| **Failed: Cannot book appointment Id of another server** |  |
| **Book Appointment** | **Failed: Patient has already book appointment in the same day with same Appointment Type** |  |
| **Failed: No appointment available for selected slot** |  |
| **Failed: Patient has already booked 3 appointments other than its server** |  |
| **Success: appointment successfully booked** |  |
| Remove appointment | **Success: Appointment is removed** |  |
| **Success: Appointment is removed with not available next appointment slot** |  |
| **Success: Appointment is removed with patient is transferred** |  |
| **Failed: No slot available** |  |
| List Appointment | **Success: All Appointment list** |  |
| Get Appointment Schedule | **Success: Empty appointment schedule** |  |
| **Success: Appointment Schedule** |  |
| Cancel Appointment | **Success: cancelled appointment** |  |
| **Failed: No record of appointment found** |  |
| Swap Appointment | **Success: Appointment swapped** |  |
| Swap Appointment | **Failed: No appointment booked of old Appointment ID** |  |
|  | **Failed: No appointment available for new appointment Id** |  |