**COMP 6231 Assignment 2 Design Documentation**

**Name**: Yajing Liu **Student ID**: 40268996

This project is to develop a distributed health care management system (DHMS) for managing medical appointments. Admins can use this system to add, list, remove, book, get, cancel, and swap appointments. And patients can use this system to book, list, cancel, swap appointments.

**Project Architecture**

A diagram of a computer

Description automatically generated with medium confidence

Figure 1 UML class diagram

As shown in Figure 1, there are 18 classes and 1 idl file in this project. The following are the details about 19 files.

**Driver**: This class provides users with command line interface and checks users’ input. It uses orb name service to get three server instances and uses server instances as parameters to create new client objects and call methods.

**Client**: This class is an abstract class handling all methods of clients.

**Patient**: This class is a subclass of Client. It calls server’s methods in bookAppointment, getAppointment, cancelAppointment and swapAppointment. It calls printInvalidCommandMessage in the remaining methods.

**Admin**: This class is a subclass of Client. It calls server’s methods in all seven methods.

**DHMS.idl**: This idl file defines module DHMSAPP and DHMS interface.

**DHMS**: This is a java interface generated by idl-to-java compiler. This interface extends DHMSOperations.

**\_DHMSStub**: This is a stub file generated by idl-to-java compiler. This class implements DHMS interface.

**DHMSHelper**: This is an abstract class generated by idl-to-java compiler.

**DHMSHolder**: This is a class generated by idl-to-java compiler.

**DHMSOperations**: This is a java interface generated by idl-to-java compiler.

**DHMSPOA**: This is an abstract class generated by idl-to-java compiler. This class implements DHMSOperations.

**Server**: This class creates three servers and binds them with proper names.

**MontrealServer**: This class is the Montreal server. It extends DHMSPOA and provides implementations for all abstract seven methods in its parent class. It uses main method to listen other servers’ requests.

**QuebecServer**: This class is the Quebec server. It extends DHMSPOA and provides implementations for all abstract seven methods in its parent class. It uses main method to listen other servers’ requests.

**SherbrookeServer**: This class is the Sherbrooke server. It extends DHMSPOA and provides implementations for all abstract seven methods in its parent class. It uses main method to listen other servers’ requests.

**ReplyAppointment**: This class extends Thread and replies to other servers’ requests about get available appointments.

**ReplyRecord**: This class extends Thread and replies to other servers’ requests about get booking records.

**Constants**: This class stores common constants used by three servers. The constants details are omitted.

**TestMultipleThreads:** This class is used to test codes when running multiple threads.

**Data Structures**

Server mainly uses two data *appointmentOuter*, *recordList, recordOtherCities*. *appointmentOuter* is a map containing another map. The inner hash map (Map <String, Integer>) represents appointment ID and capacity. The outer hash map (Map <String, Map <String, Integer>>) represents appointment type and inner map. *recordList* is a string linked list (List <String>). *recordList* stores booking records in the format “patientID appointmentID appointmentType”. *recordOtherCities* is similar to *recordList* but it only stores booking records that patients’ cities are different from appointments’ cities. *recordOtherCities* is used to check whether the patient books more than three appointments in other cities. Apart from data parameters stored in server objects, I also use txt to store the appointments and records to response other cities’ requests for getting these data.

**Major Techniques**

**CORBA**

I used CORBA for client-server communication. I defined module and interface in DHMS.idl (Figure 2). I generated DHMSApp folder by using idl-to-java complier (Figure 3). I created three server objects and bound them with proper names in Server class (Figure 4). I used name service to get server instances and called servers’ methods in Driver class (Figure 5).

A screen shot of a computer code

Description automatically generated

Figure 2 DHMS.idl

A screenshot of a computer

Description automatically generated

Figure 3 DHMSApp folder

A computer screen shot of a program

Description automatically generated

Figure 4 Server class

A screen shot of a computer program

Description automatically generated

Figure 5 Driver class

**UDP/IP**

I used UDP/IP to handle server-server communication. Server can send requests to get other servers’ data (appointments or booking records) and to book or cancel appointments in other cities (e.g. Figure 6). And other servers can listen to requests and give responses (Figure 7). In order to reply different requests parallelly, servers use different threads (RelplyAppointment and ReplyRecord).

A screen shot of a computer program

Description automatically generated

Figure 6 Send UDP request

A computer screen shot of a program

Description automatically generated

Figure 7 Receive UDP requests and send responses

**Multiple threads**

I used several methods to handle the multiple threads for server methods including synchronized data types (synchronizedMap, synchronizedList) (Figure 8) and ReadWriteLock (writelock and readlock) (Figure 9).

A black background with white text

Description automatically generated

Figure 8 synchronizedMap and synchronizedList

A screen shot of a computer

Description automatically generated

Figure 9 writelock

**Test Scenarios**

1. **Description**: a patient cannot perform any admin operations. **Input**: (1) MTLP0001 (2) addAppointment MTLA080224 Physician 4 **Output**: Figure 9 **Pass**

A black screen with white text

Description automatically generated

Figure 10 Patient cannot perform any admin operations

2. **Description**: an admin can add an appointment. **Input**: addAppointment MTLA080224 Physician 2 **Output**: Figure 10 **Pass**

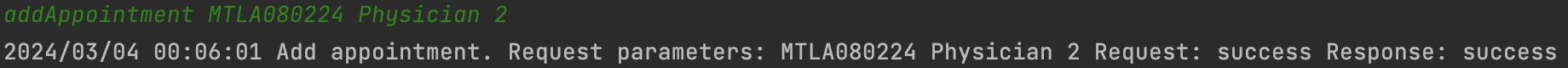


Figure 11 Admin can add appointment

3. **Description**: an admin cannot add same appointments twice. **Input**: (1) addAppointment MTLA080224 Physician 2 (2) addAppointment MTLA080224 Physician 2 **Output**: Figure 11 **Pass**

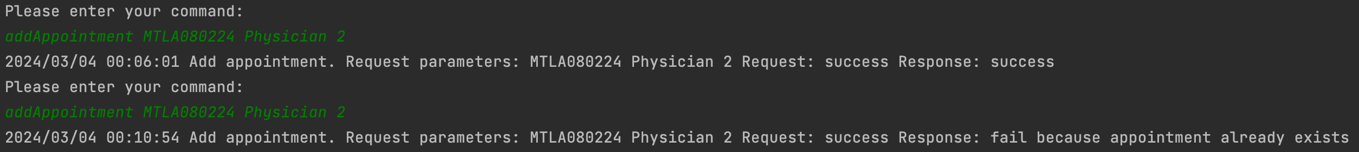


Figure 12 Admin cannot add same appointments twice

4. **Description**: an admin can remove appointment. **Pre-conditions**: Montreal server has appointment (ID: MTLE080224, type: Surgeon, capacity: 3). **Input**: removeAppointment MTLE080224 Surgeon **Output**: Figure 12 **Pass**



Figure 13 Admin can remove appointment

5. **Description**: an admin cannot remove a non-exist appointment. **Input**: removeAppointment MTLA080224 Surgeon **Output**: Figure 13 **Pass**



Figure 14 Admin cannot remove a non-exist appointment

6. **Description**: if an appointment is booked by a patient, admin will delete the appointment and book the next available appointment for the patient. **Pre-conditions**: (1) Montreal server has two appointments (ID: MTLA080224, type: Physician, capacity: 2; ID: MTLM080224, type: Physician, capacity: 2). (2) Patient (MTLP0001) books an appointment (ID: MTLM080224, type: Physician) **Input**: removeAppointment MTLM080224 Physician **Output**: Figure 14 **Pass**

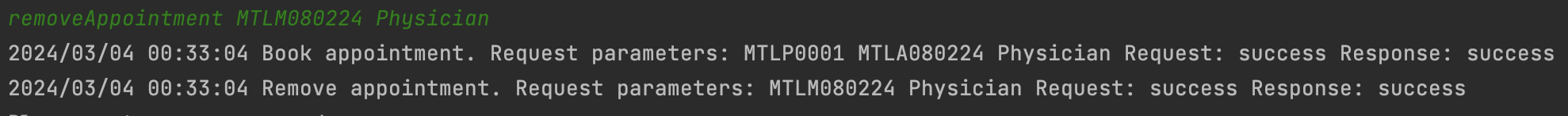


Figure 15 Delete a booked appointment

7. **Description**: admin can list appointments in three cities. **Pre-conditions**: (1) Montreal server has an appointment (ID: MTLA080224, type: Physician, capacity: 2). (2) Quebec server has an appointment (ID: QUEA100224, type: Physician, capacity: 2). (3) Sherbrooke server has an appointment (ID: SHEA090224, type: Physician, capacity: 2). **Input**: listAppointmentAvailability Physician **Output**: Figure 15 **Pass**



Figure 16 List appointment availability

8. **Description**: a patient can book an appointment **Pre-conditions**: Montreal server has an appointment (ID: MTLA080224, type: Physician, capacity: 2). **Input**: bookAppointment MTLP0001 MTLA080224 Physician **Output**: Figure 16 **Pass**



Figure 17 Book appointment

9. **Description**: a patient cannot book a same type of appointments multiple times in a day **Pre-conditions**: (1) Montreal server has an appointment (ID: MTLA080224, type: Physician, capacity: 2; ID: MTLM080224, type: Physician, capacity: 2). (2) Patient (MTLP0001) books an appointment (ID: MTLM080224, type: Physician) **Input**: bookAppointment MTLP0001 MTLA080224 Physician **Output**: Figure 17 **Pass**



Figure 18 Patient cannot book a same type of appointments multiple times in a day

10. **Description**: a patient can book an appointment in other cities **Pre-conditions**: Quebec server has an appointment (ID: QUEA100224, type: Physician, capacity: 2). **Input**: bookAppointment MTLP0002 QUEA100224 Physician **Output**: Figure 18 **Pass**



Figure 19 Book appointment in other cities

11. **Description**: a patient cannot book more than three appointments in other cities in a week. **Pre-conditions**: (1) Quebec server has four appointments (ID: QUEA100224, type: Physician, capacity: 2; ID: QUEA100224, type: Physician, capacity: 2; ID: QUEA110224, type: Physician, capacity: 2; ID: QUEA120224, type: Physician, capacity: 2; ID: QUEA130224, type: Physician, capacity: 2;). (2) Patient (MTLP0002) books appointments (ID: QUEA100224, type: Physician; ID: QUEA110224, type: Physician; ID: QUEA120224, type: Physician) **Input**: bookAppointment MTLP0002 QUEA130224 Physician **Output**: Figure 19 **Pass**



Figure 20 Patient cannot book more than three appointments in other cities in a week

12. **Description**: a patient can get appointment schedule **Pre-conditions**: (1) Quebec server has an appointment (ID: QUEA100224, type: Physician, capacity: 2). (2) Patient (MTLP0002) books appointments (ID: QUEA100224, type: Physician) **Input**: getAppointmentSchedule MTLP0002 **Output**: Figure 20 **Pass**



Figure 21 Get appointment schedule

13. **Description**: a patient can cancel appointment **Pre-conditions**: (1) Montreal server has an appointment (ID: MTLA140224, type: Dental, capacity: 4). (2) Patient (MTLP0002) books appointments (ID: MTLA140224, type Dental) **Input**: cancelAppointment MTLP0002 MTLA140224 **Output**: Figure 21 **Pass**

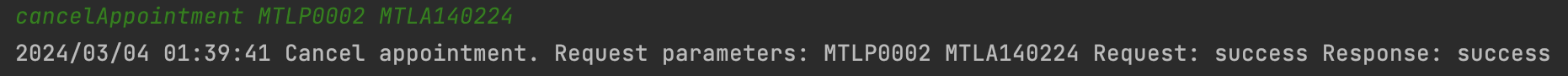


Figure 22 Cancel appointment

14. **Description**: a patient can cancel appointment in other cities **Pre-conditions**: (1) Quebec server has an appointment (ID: QUEA100224, type: Physician, capacity: 2). (2) Patient (MTLP0002) books appointments (ID: QUEA100224, type: Physician) **Input**: cancelAppointment MTLP0002 QUEA100224 **Output**: Figure 22 **Pass**

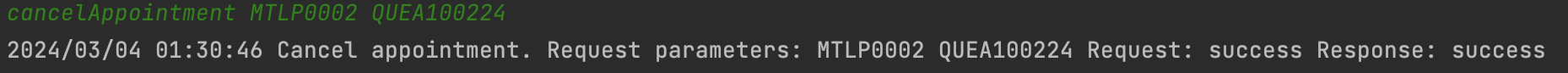


Figure 23 Cancel appointment in other cities

15. **Description**: a patient cannot cancel a non-exist appointment **Input**: cancelAppointment MTLP0002 MTLE080224 **Output**: Figure 23 **Pass**

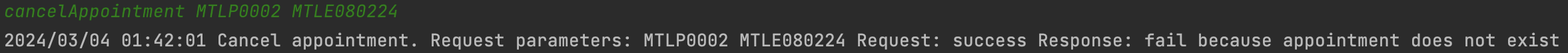


Figure 24 Patient cannot cancel a nonexist appointment

16. **Description**: a patient can swap an appointment in other cities **Pre-conditions**: (1) Montreal server has an appointment (ID: MTLA080224, type: Physician, capacity: 2). (2) Quebec server has an appointment (ID: QUEA100224, type: Physician, capacity: 2). (3) Patient (MTLP0001) books an appointment (ID: MTLA080224, type: Physician). **Input**: swapAppointment MTLP0001 MTLA080224 Physician QUEA100224 Physician **Output**: Figure 24 **Pass**

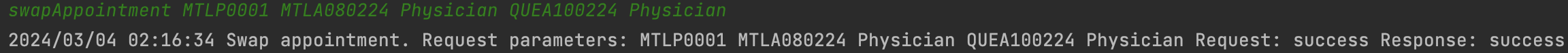


Figure 25 Swap appointment

17. **Description**: a patient cannot swap an appointment if the patient does not book the old appointment **Pre-conditions**: (1) Montreal server has an appointment (ID: MTLA080224, type: Physician, capacity: 2). (2) Quebec server has an appointment (ID: QUEA100224, type: Physician, capacity: 2). **Input**: swapAppointment MTLP0001 MTLA080224 Physician QUEA100224 Physician **Output**: Figure 25 **Pass**



Figure 26 Patient cannot swap appointment if the patient does not book the old appointment

18. **Description**: a patient cannot swap an appointment if the new appointment has no available slot. **Pre-conditions**: (1) Montreal server has an appointment (ID: MTLA080224, type: Physician, capacity: 2). (2) Quebec server has an appointment (ID: QUEA100224, type: Physician, capacity: 1). (3) Patient (QUEP0001) books appointment (ID: QUEA100224, type: Physician) (4) Patient (MTLP0001) books appointment (ID: MTLA080224, type: Physician). **Input**: swapAppointment MTLP0001 MTLA080224 Physician QUEA100224 Physician **Output**: Figure 26 **Pass**



Figure 27 Patient cannot swap an appointment if the new appointment has no available slot

19. **Description**: server can provide proper synchronization for multiple clients. **Pre-conditions**: In TestMultipleThreads class, patientMTL try to swap appointment and patientQUE try to book the same appointments at the same time. Before providing proper synchronization, patientMTL cancels old appointments successfully but books new appointment unsuccessfully since patientQUE books the same appointments successfully after cancelling and before booking a new one. **Output**: After providing synchronization, patientMTL will not swap appointment (Figure 28) **Pass**

****

Figure 28 Synchronization results

**Most difficult parts**

(1) Build CORBA connections between server and client. (2) Solve the stuck responses caused by waiting other requests in a not-parallel multiple different UDP connections.