Computer Science and Software Engineering Département

COMP6231/1 – DISTRIBUTED SYSTEMS DESIGN

Summer 2018

Assignment 1

Team:

First Name	Last Name	Student ID	Email ID
Ezhilmani	Aakaash	40071067	aakaash07@gmail.com
Akhila	Chilukuri	40071241	akhilachilukuri1@gmail.com
Vigneswar	Mourouguessin	40057918	imvigneswar@gmail.com
Vaishnavi	Venkatraj	40049798	whyshu@gmail.com

TABLE OF CONTENTS

A. Overview: 3

B. Description: 4

C. Architecture: 5

D. Test Scenarios: 7

E. Challenges Faced : 9

E. UML diagrams : 9

F. References : 11

A. Overview:

The Distributed Class Management System, is a distributed system used by center managers to manage information regarding the teachers and students across different centers. The DCMS is used by the Manager's to maintain Teacher and Student Record collectively with respect to three centers locations: Montreal (MTL), Laval (LVL) and Dollard-des Ormeaux (DDO)

This system also provides the manager with a set of functions that can be performed on the Teacher and the Student data which helps the manager perform his responsibilities.

Teacher Record contains the following fields:

1. First Name
2. Last Name
3. Address
4. Phone
5. Specialization
6. Location.

Student Record contains the following fields:

1. First Name
2. Last Name
3. Courses Registered
4. Status
5. Status Date

The system provides the following functionalities/operations:

1) Create Teacher Record (createTRecord)
This method is used to store details of a Teacher into the system and assign a unique ID to every Teacher in the system.

2) Create Student Record (createSRecord)
This method is used to store details of a Student into the system and assign a unique Student ID to every Student in the system.

3) Get Record Count (getRecordCounts)
This method is used to get the count of the records from all the three servers across the system - Montreal (MTL), Laval (LVL) and Dollard-des Ormeaux (DDO), using UDP Sockets and return the actual data to the Client (Manager).

4) Edit Record (editRecord)
This method is used to edit the existing records in the Teacher and Student record i.e. records created via methods createTRecord() and createSRecord(). Upon success or failure it returns a message to the manager and the logs are updated with this information.

The fields that should be allowed to change are

- Teacher Record - address, phone and location
- Student Record - course registered, status and status date

B. Description:

Java Remote Method Invocation :

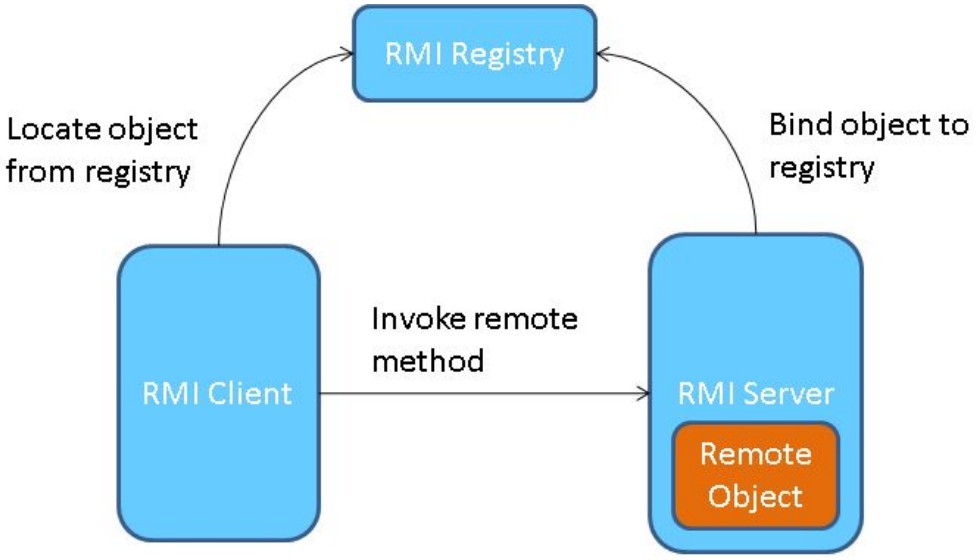
The Distributed Class Management System is designed on functionality of the Java RMI (Remote Method Invocation), which is a mechanism that allows one Java Virtual Machine (JVM) running object to invoke methods on an object running in another JVM. It facilitates the remote calling of Java object methods and sharing of resources and services.

Basics of an RMI Application

In an RMI application, we have two programs, a server program and a client program

Server program : It resided in the server, inside the server program, a remote object is created and reference of that object is made available for the client (using the registr**y).**

Client program : It resides in the client. The client program requests the remote objects on the server and tries to invoke its methods.



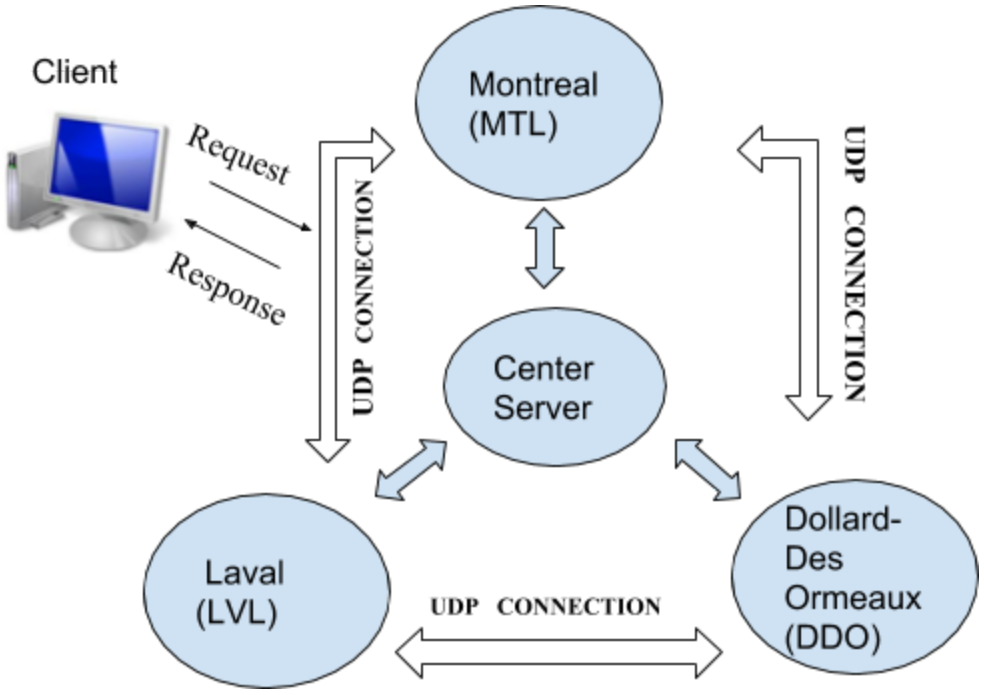
C. Architecture:

In DCMS, we have Managers for each of the different locations who would be acting as the Clients in the System. The Manager (Montreal / Laval / Dollard-Des-Ormeaux) have access to all the functionalities in the system. A Manager can add a Teacher Record and based on the information provided by the manager the createTRecord() method is called and the data is stored in an Hashmap as per the requirement and simultaneously all the events and actions are captured in the Log File as well.

Similarly, the manager also has control to enter a Student Record in the system by calling the createSRecord() method of the system and a new student is being registered in the system and logs of the same are also stored at the appropriate file and location.

There is also a functionality that helps to get the Record Count across locations which would be a total of Teacher and Student Record. This functionality has been implemented using the UDP connectivity across all the available servers and return the count of records on that particular server.

The below architecture represents the Client Server Communication and the connectivity established between servers via UDP Socket and how the request from the client goes to the central server and the appropriate response that is processed and sent back to the client.



Another functionality that the manager has is to edit the data record which he has entered into the system whether it be a Teacher Record or a Student Record. The Manager is allowed has to provide a valid Record ID that he has to edit. After which the program has to be provided with the field name which is to be edited and the new value that has to be replace instead of a new value. This would allow the manager to have the flexibility of managing the data effectively as and when required.

All the teacher records and students servers are stored in the Hashmap of the corresponding manager ID. The Records are placed in several lists that are stored in a hashmap according to the first letter of the last name indicated in the records.

D. Test Scenarios:

Test ID	Test Description	Expected Result	Actual Result
T1	Manager requests for entering data into the Teacher Record.	The record should be successfully placed in the hashmap.	The Record was inserted in the hashmap successfully
T2	Client requests for entering data into the Student Record.	The record should be successfully placed in the hashmap.	The Record was inserted in the hashmap successfully
T3	Client should be able to successfully login to the system with Valid Credentials. Test Data: (MTLXXXX / LVLXXXX / DDOXXX)	The Client should be able successfully login into the system with Valid Credentials.	The Client was able to login to the system with valid credentials.
T4	Client should not able to login to the system with Invalid Credentials.	The system should not allow to login to the system with invalid credentials.	The system did not allow to login with invalid credentials.
T5	Client should get Record Count across all servers by using the getRecord Count function of the system.	The client should be displayed the count of all the records on the available servers.	The client was displayed the count of all the records on the available servers.
T6	Client should be able to edit the Record when provided with Valid Record ID	The Client should be able to successfully edit the Record when provided with Valid Record ID	The Client was able to successfully edit the Record when provided with Valid Record ID

T7	Checking for Phone Number validation in Teacher Record.	The system should not allow any input other that numeric values for Phone Number field of the Teacher Record.	The system does not allow any input other that numeric values for Phone Number field of the Teacher Record.
T8	Checking for Status field validation in Student Record.	The system should not allow any values other that (Active/Inactive) for Status field of the Student Record.	The system does not allow any values other that (Active/Inactive) for Status field of the Student Record.
T9	Validating Input for Location field in the Edit Record.	The user should be able to enter only 3 values (MTL/LVL/DDO) else system should throw an error.	The user was able to enter only 3 values (MTL/LVL/DDO) else system is throwing an error.
T10	The user should not be allowed to edit any fields other than Address, Phone and Location for the Teacher Record.	The system should not allow editing of any field except Address, Phone and Location.	The system did not allow editing of any field except Address, Phone and Location.
T11	The user should not be allowed to edit any fields other than Course Registered, Status and Status Date for the Student Record.	The system should not allow editing of any field except Course Registered, Status and Status Date.	The system did not allow editing of any field except Course Registered, Status and Status Date.
T12	Checking whether record count function returns records in the all the server including the records which has lastname starting with the same letter.	The system should return the total count of the records in the server	The system did return the total count of the all the records in the server.

E. Challenges Faced :

- **Client-Server RMI architecture implementation:** Understanding RMI and implementing the basic architecture with all the methods was also time-consuming.

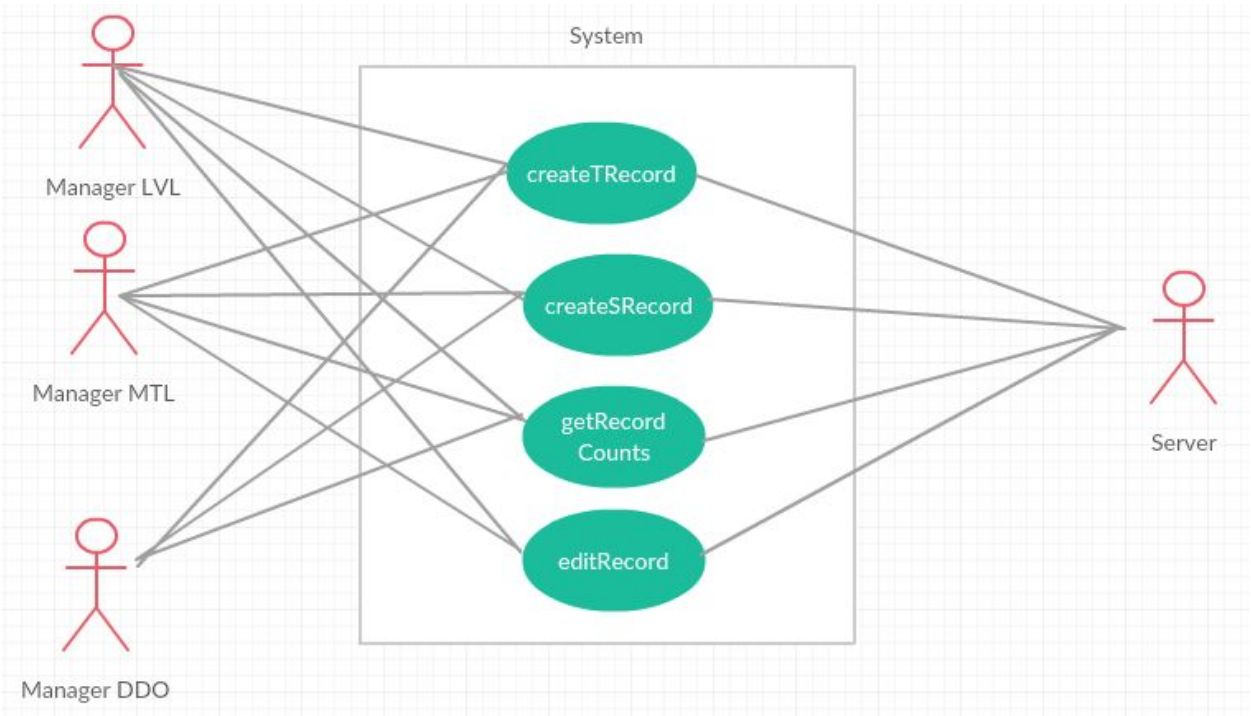
- **UDP Connection establishment :** Establishing UDP connection among the 3 servers was time-consuming and challenging.

- **Get Record Count Implementation :** Getting the record count of three servers and communicating accordingly was also challenging.

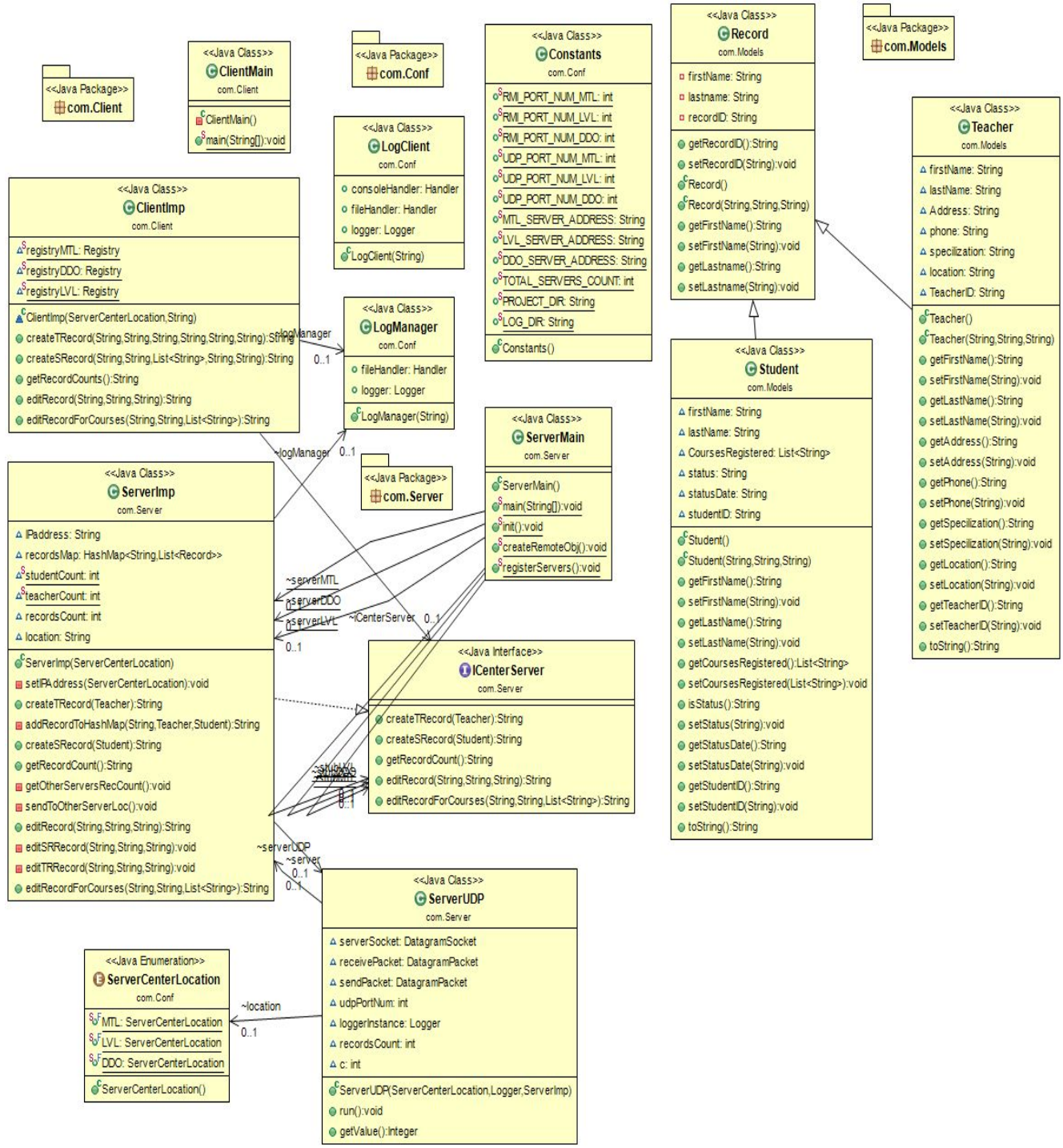
- **Edit Record Implementation:** Editing the records and updating appropriate server was challenging.

E. UML diagrams :

Use Case Diagram



Class Diagram


<<Java Package>>
com.Client
<<Java Class>>
ClientMain
com.Client
ClientMain()
main(String[]):void
<<Java Package>>
com.Conf
<<Java Class>>
Constants
com.Conf
RMI_PORT_NUM_MTL: int
RMI_PORT_NUM_LVL: int
RMI_PORT_NUM_DDO: int
UDP_PORT_NUM_MTL: int
UDP_PORT_NUM_LVL: int
UDP_PORT_NUM_DDO: int
MTL_SERVER_ADDRESS: String
LVL_SERVER_ADDRESS: String
DDO_SERVER_ADDRESS: String
TOTAL_SERVERS_COUNT: int
PROJECT_DIR: String
LOG_DIR: String
Constants()
<<Java Class>>
Record
com.Models
firstName: String
lastname: String
recordID: String
getRecordID():String
setRecordID(String):void
Record()
Record(String,String,String)
getFirstName():String
setFirstName(String):void
getLastname():String
setLastname(String):void
<<Java Package>>
com.Models
<<Java Class>>
Teacher
com.Models
firstName: String
lastName: String
Address: String
phone: String
specilization: String
location: String
TeacherID: String
Teacher()
Teacher(String,String,String)
getFirstName():String
setFirstName(String):void
getLastName():String
setLastName(String):void
getAddress():String
setAddress(String):void
getPhone():String
setPhone(String):void
getSpecilization():String
setSpecilization(String):void
getLocation():String
setLocation(String):void
getTeacherID():String
setTeacherID(String):void
toString():String
<<Java Class>>
Student
com.Models
firstName: String
lastName: String
CoursesRegistered: List<String>
status: String
statusDate: String
studentID: String
Student()
Student(String,String,String)
getFirstName():String
setFirstName(String):void
getLastName():String
setLastName(String):void
getCoursesRegistered():List<String>
setCoursesRegistered(List<String>):void
isStatus():String
setStatus(String):void
getStatusDate():String
setStatusDate(String):void
getStudentID():String
setStudentID(String):void
toString():String
<<Java Package>>
com.Client
<<Java Class>>
ClientImp
com.Client
registryMTL: Registry
registryDDO: Registry
registryLVL: Registry
ClientImp(ServerCenterLocation,String)
createTRecord(String,String,String,String,String,String):String
createSRecord(String,String,List<String>,String,String):String
getRecordCounts():String
editRecord(String,String,String):String
editRecordForCourses(String,String,List<String>):String
<<Java Class>>
LogClient
com.Conf
consoleHandler: Handler
fileHandler: Handler
logger: Logger
LogClient(String)
<<Java Class>>
LogManager
com.Conf
fileHandler: Handler
logger: Logger
LogManager(String)
~logManager
0..1
~logManager
0..1
<<Java Class>>
ServerMain
com.Server
ServerMain()
main(String[]):void
init():void
createRemoteObj():void
registerServers():void
<<Java Package>>
com.Server
<<Java Class>>
ServerImp
com.Server
IPaddress: String
recordsMap: HashMap<String,List<Record>>
studentCount: int
teacherCount: int
recordsCount: int
location: String
ServerImp(ServerCenterLocation)
setIPAddress(ServerCenterLocation):void
createTRecord(Teacher):String
addRecordToHashMap(String,Teacher,Student):String
createSRecord(Student):String
getRecordCount():String
getOtherServersRecCount():void
sendToOtherServerLoc():void
editRecord(String,String,String):String
editSRRecord(String,String,String):void
editTRRecord(String,String,String):void
editRecordForCourses(String,String,List<String>):String
~serverMTL
~serverDDO
~serverLVL
~ICenterServer
0..1
0..1
<<Java Interface>>
ICenterServer
com.Server
createTRecord(Teacher):String
createSRecord(Student):String
getRecordCount():String
editRecord(String,String,String):String
editRecordForCourses(String,String,List<String>):String
~serverUDP
~server
0..1
0..1
<<Java Class>>
ServerUDP
com.Server
serverSocket: DatagramSocket
receivePacket: DatagramPacket
sendPacket: DatagramPacket
udpPortNum: int
loggerInstance: Logger
recordsCount: int
c: int
ServerUDP(ServerCenterLocation,Logger,ServerImp)
run():void
getValue():Integer
<<Java Enumeration>>
ServerCenterLocation
com.Conf
MTL: ServerCenterLocation
LVL: ServerCenterLocation
DDO: ServerCenterLocation
ServerCenterLocation()
~location
0..1

F. References :

1. **http://www.sourcetricks.com/2014/10/java-rmi.html**
2. **www.google.com**
3. **https://docs.oracle.com/javase/tutorial/rmi/server.html**
4. **https://docs.oracle.com/javase/tutorial/rmi/**
5. **https://www.javatpoint.com/RMI**
6. **http://www.baeldung.com/udp-in-java**
7. **https://systembash.com/a-simple-java-udp-server-and-udp-client/**
8. **https://www.loggly.com/ultimate-guide/java-logging-basics/**
9. **http://www.objectaid.com/**
10. **https://creately.com**