Computer Science and Software Engineering Department

COMP6231/1 – DISTRIBUTED SYSTEMS DESIGN

Assignment 2

Team:

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

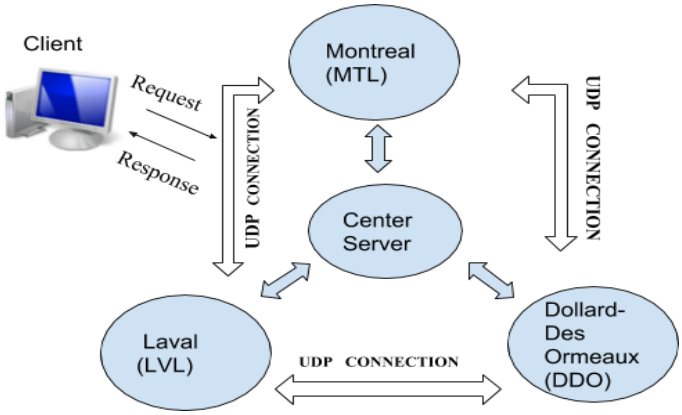
CONTENTS

OVERVIEW:

The Distributed Class Management System, is a distributed system that performs various functionalities to maintain information across servers while maintaining concurrency by synchronizing the functionalities. This is to make sure that when the functions are executed across multiples servers, the actions performed do no not cause error or interfere with other processes. To do so, UDP communication has been established between the servers.

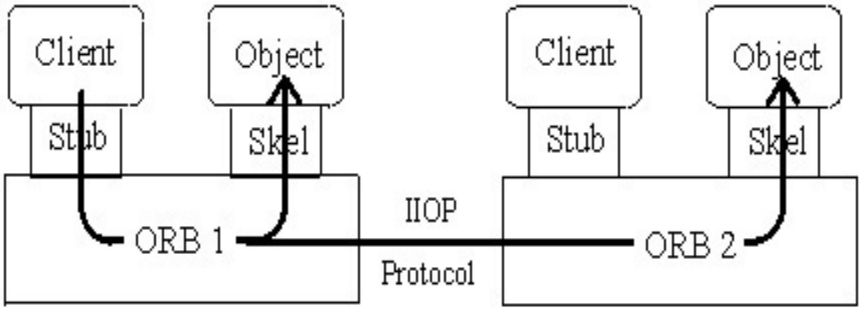
ARCHITECTURE:

In DCMS, the Managers in each of the different locations would be acting as the Clients in the System. The Managers are each assigned with a unique manager ID with which they communicate with Servers to perform the necessary operations. There are Servers in 3 locations MTL, LVL and DDO. The Client and Server communicate using the **Common Object Request Broker Architecture** (**CORBA**) which is the key implementation of this assignment. The Clients and Server maintain separate Logs of the actions performed.

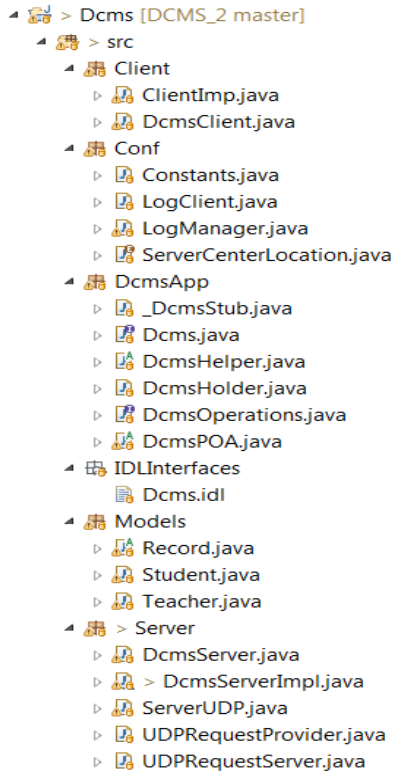


[Wikipedia] The CORBA specification dictates there shall be an ORB through which an application would interact with other objects. This is how it is implemented in practice.

1. The application simply initializes the ORB, and accesses an internal *Object Adapter*, which maintains things like reference counting, object (and reference) instantiation policies, and object lifetime policies.
2. The Object Adapter is used to register instances of the *generated code classes*. Generated code classes are the result of compiling the user IDL code, which translates the high-level interface definition into an OS- and language-specific class base for use by the user application. This step is necessary in order to enforce CORBA semantics and provide a clean user process for interfacing with the CORBA infrastructure.

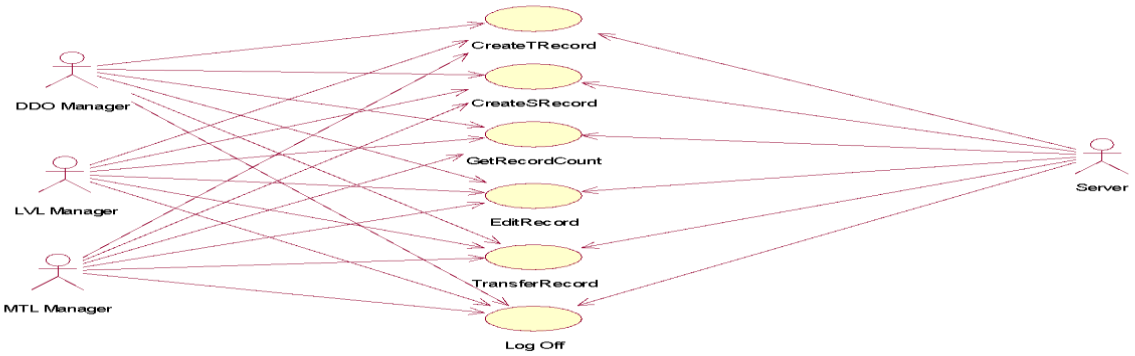


PROJECT STRUCTURE:

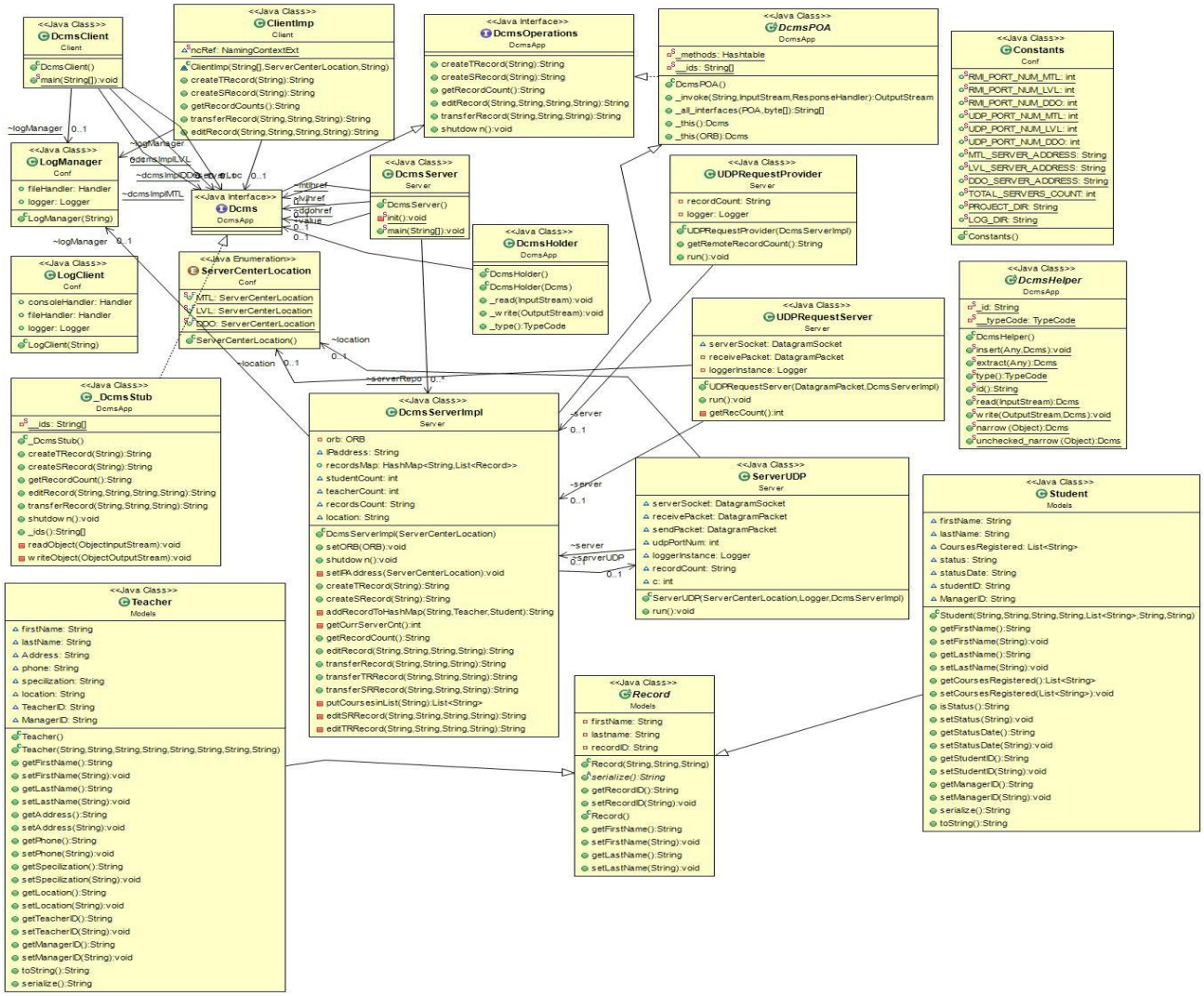


The Use-case diagram and the Class diagrams of the implementation are illustrated below:

USE-CASE DIAGRAM:



CLASS DIAGRAM:



UDP SERVER COMMUNICATION:

[Wikipedia] UDP uses a simple connectionless communication model with a minimum of protocol mechanism. UDP provides checksums for data integrity, and port numbers for addressing different functions at the source and destination of the datagram.

The Servers in the 3 locations communicate using this protocol for serving the client's Get Record Count operation.

IMPLEMENTATION OF CONCURRENCY & SYNCHRONIZATION:

[Wikipedia] Thread synchronization is defined as a mechanism which ensures that two or more concurrent processes or threads do not simultaneously execute some particular program segment known as critical section. Processes' access to critical section is controlled by using synchronization techniques. When one thread starts executing the critical section (serialized segment of the program) the other thread should wait until the first thread finishes. If proper synchronization techniques are not applied, it may cause a race condition where the values of variables may be unpredictable and vary depending on the timings of context switches of the processes or threads.

The DCMS implementation needs to be synchronized as multiple clients communicate with a server in the same location. We have implemented synchronization with the synchronized functionality that's available in JAVA. The server methods are all synchronized and the methods that are used to communicate from clients are all thread safe in both client-server communication and server-server communication.

OPERATIONS:

1. **CREATE TEACHER RECORD**

   Stores the details of the Teacher into the server's hash map and assign a unique ID to every Teacher in the system.

2. **CREATE STUDENT RECORD**

   Stores the details of the Student into the server's hash map and assign a unique Student ID to every Student in the system.

3. **GET RECORD COUNT**

   Gets the count of the records from all the three servers across the system- Montreal (MTL), Laval (LVL) and Dollard-des Ormeaux (DDO), using UDP communication and return the

actual data to the Client (Manager).

4. **EDIT RECORD**

Edits the existing records in the servers' hashmaps 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.

5. **TRANSFER RECORD**

This method, transfers the record among the servers, by deleting the record in the current server's hashmap and adding the record to remote server.

CHALLENGES FACED:

Using CORBA Architecture:

Implementing the CORBA architecture and understanding the basic functionality of CORBA was challenging.

Maintaining Concurrency:

Synchronizing the right functions to regulate the concurrent access took some understanding of the working.

UDP Connection establishment:

Establishing UDP connection among the 3 servers was time-consuming and challenging.

Transfer Record Implementation:

Transferring the records among the three servers and communicating accordingly was also challenging.

Altering Previous Code:

IDL interface implementation involved many data type changes in the existing server code.

TEST CASES:

Test ID	Test Description	Expected Result	Actual Result
T1	Client requests to create a record and Insert teacher record to server's hashmap	Record should be inserted into hashmap.	Record was inserted into hashmap successfully.
T2	Client requests to create a record and Insert student record to server's hashmap	Record should be inserted into hashmap.	Record was inserted into hashmap successfully.
T3	Login using valid credentials. Test Data: (MTLXXXX / LVLXXXX / DDOXXX)	Successful login.	Yes, logged in.
T4	Login using invalid credentials.	Unable to login.	Yes, unable to login.
T5	Get record count from all servers.	Returns the record count to all the servers.	Yes, successfully gets the record count
T6	Edit SR/TR record, with valid student ID.	Successfully edited the record.	Yes, successfully edited the record.

T8	Validation of Phone number, status, location, Courses registered.	Request .	The system does not allow any values other that (Active/Inactive) for Status field of the Student Record.
T9	Validating input for Location field while editing record.	The user should be able to enter only 3 values (MTL/LVL/DDO)	The user was able to enter only 3 values (MTL/LVL/DDO)
T10	Edit Teacher record fields	Edit the appropriate record and return success	Record was edited and the fields were updated, returned success.
T11	Edit Student record fields	Edit the appropriate record and return success.	Record was edited, and the fields were updated, returned success.
T12	Get the record count from all servers.	Return server name, record count to the client.	The record count from all servers were obtained successfully and returns success.
T13	Transfer record to itself	The server should deny the process.	System made sure that the transfer did not happen.

T14	Transfer record from one server to another server.	Record should be transferred from the source to destination, and removed from the source.	Record was transferred from the source to destination, and removed from the source.
T15	Multiple clients request to same server.	Serve the request to the appropriate client.	The server communicated and returned the result to the appropriate client.
T16	Multiple UDP requests at the same time – getreccount, transferrecord	The UDP server should serve the appropriate packet and return the correct result.	The UDP server served the appropriate packet and return the correct result.

CONCLUSION:

The CORBA Client-Server architecture was implemented successfully, and the necessary operations were implemented, executed and tested successfully.

REFERENCES:

https://docs.oracle.com/javase/7/docs/technotes/guides/idl/jidlExample.html

https://en.wikipedia.org/wiki/Common_Object_Request_Broker_Architecture

https://docs.oracle.com/javase/7/docs/technotes/guides/idl/corba.html

http://www.oracle.com/technetwork/articles/javase/rmi-corba-136641.html

https://www.geeksforgeeks.org/synchronized-in-java/