UNIVERSITÉ Concordia UNIVERSITY

Computer Science and Software Engineering Département

COMP 6231/1 – DISTRIBUTED SYSTEMS DESIGN

Summer 2018

Assignment III

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

OVERVIEW

The Web Service implementation of **Distributed Class Management System** (DCMS) strives to provide the manager (User) with the ability to create and maintain student and teacher's records. This implementation helps the manager perform his/her duties in a concurrent and synchronous fashion. This is achieved with the help of Web Services, which is basically a medium of communication between the user and client carried out using strict protocols. Web Services are developed based on the present technologies such as **SOAP**, Web Services Description Language (**WSDL**), Extensible Markup Language (**XML**), Hypertext Transfer Protocol (**HTTP**), Universal Description, Discovery, and Integration (**UDDI**).

Web Service - Definition and Usage:

As mentioned before, Web Services provide means of communication between various servers and clients located all around the globe. More specifically, a Web service is a software application with a standardized way of providing interoperability between disparate applications. This is done with the help of technologies such as SOAP, WSDL, UDDI etc..

In general, a web service is used for machine to machine communication that results in providing the needed interface for the user. This includes human readable documents in the form of XML docs.

Web Services can be broadly categorized into two.
- SOAP-Based
- RESTful Services

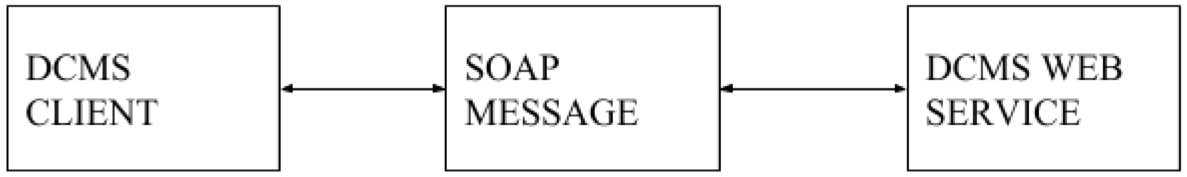
Usage:

Web Services can be defined in terms or server-client communication. Hence, a server program that is coded onto the server and a client code that resides with the client side is established.

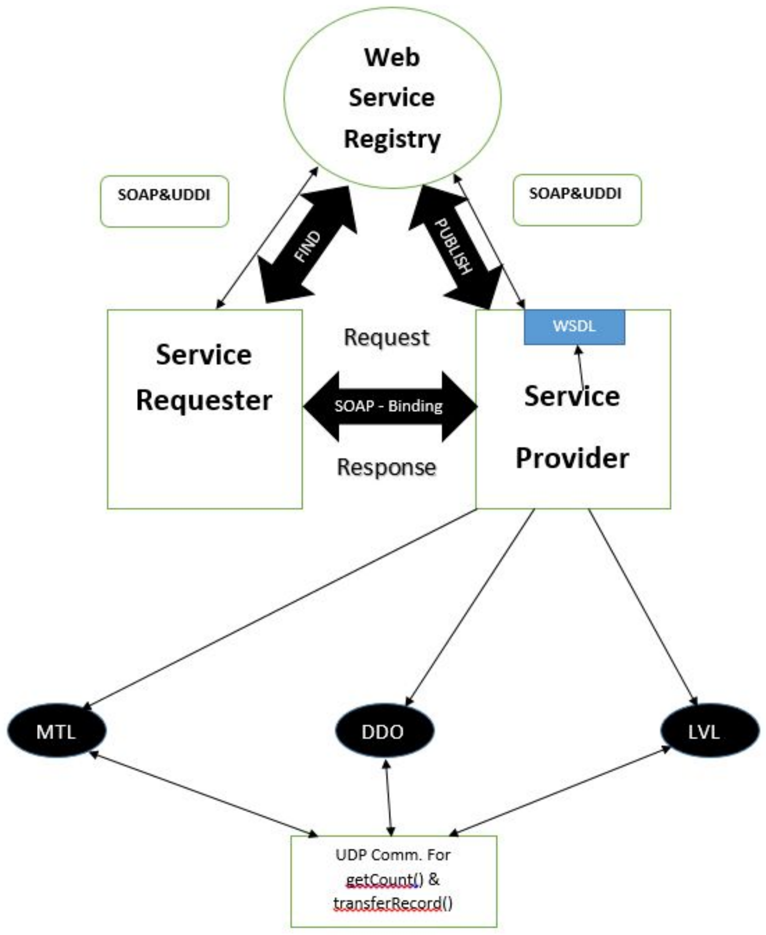
- The entire service is documented in term of XML that provides operations, messages, and bindings along with a optional URL to call for.
- The structure of this service is defined in the XML Schema referred to be the WSDL.
- Keeping the standards in mind, the server and client code is written along with the needed functions.
- Once all of this is done, the service is then published which maybe access or used by a client upon request.

WEB-SERVICES IMPLEMENTATION & TECHNIQUES INVOLVED:

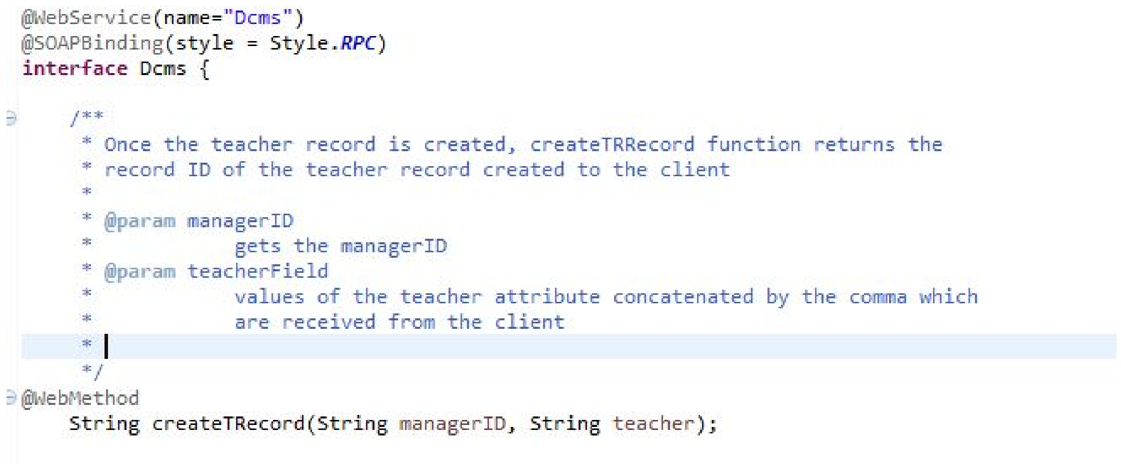
To implement three servers as individual web services, the project is created as a Dynamic Web project, where the Target runtime could be specified as any popular web server, for our use case we have chosen Tomcat v9.0 and all other web module versions are specified as needed or left with the default values. The Source folder are created in the build path, which will have the major client and server code.



ARCHITECTURE:



A **service endpoint interface** (SEI) is a Java interface or class, respectively, that declares the methods that a client can invoke on the service. The web service implementation class implicitly defines an SEI.We provided an interface that defines the public methods made available in the endpoint implementation class.


```
@WebService(name="Dcms")
@SOAPBinding(style = Style.RPC)
interface Dcms {

    /**
     * Once the teacher record is created, createTRRecord function returns the
     * record ID of the teacher record created to the client
     *
     * @param managerID
     *            gets the managerID
     * @param teacherField
     *            values of the teacher attribute concatenated by the comma which
     *            are received from the client
     *
     */
@WebMethod
    String createTRecord(String managerID, String teacher);
```

The service implementation class, Dcms, is annotated as a web service endpoint using the **@WebService** annotation, declares 5 methods annotated with the **@WebMethod** annotation, which exposes the annotated method to the web service clients

1. Create Teacher Record (CreateTRecord)

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 (CreateSRecord)

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 (getRecordCount)

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 (editRecord)

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 (TransferRec)

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.

IMPORTANT PART OF WEB SERVICE IMPLEMENTATION :

The DCMS server implementation class of the web services is annotated in the following way,

@WebService(endpointInterface = "Server.Dcms", portName = "DcmsPort", serviceName = "DcmsService")

The DCMS Client class is a stand-alone application client that accesses the 5 functionalities of DCMS WebService. This call is made through a port, a local object that acts as a proxy for the remote service. The port is created at development time by the wsimport task, which generates JAX-WS portable artifacts based on a WSDL file.

eclipse_workspace\Dcms_webservice\DcmsService.wsdl

The web services' wsdl files can be viewed using the below URLs , after the client server code is started.

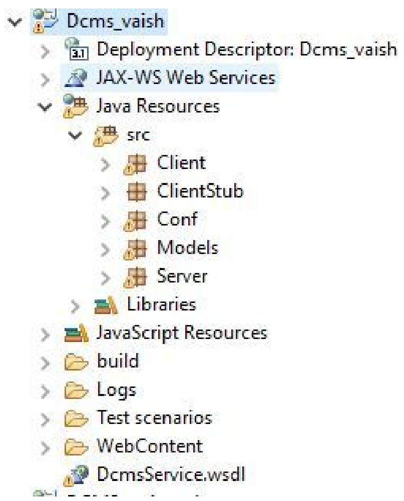
http://<MTL_SERVER_IP_ADDRESS>:3333/MTL?wsdl

http://<LVL_SERVER_IP_ADDRESS>:5555/LVL?wsdl

http://<DDO_SERVER_IP_ADDRESS>:4444/DDO?wsdl

<u>**SEPARATION OF CLIENT & SERVER :**</u>

The client and server code are under different packages and run independent of each other, methods in server, which are shared among multiple classes in server only, are kept package specific, the project structure is shown below



The DCMS contains managers that access the servers at 3 different locations (Montreal / Laval / Dollard-Des-Ormeaux).

The client managers are served with 5 functionalities which are explained more in detail below:

'**createTRecord()**' method is one of them, wherein the manager has the ability to create and add a Teacher record based on the input given. This is then put into the data structure (server's Hashmap) that is being used. All of the actions performed are kept track of by logging them in separate log folders for all 3 locations. Similarly, the managers can also create and add a Student record based on the input data using '**createSRecord()'** method.

Another functionality is the '**editRecord()**' method. As the name suggests, this method allows the manager to **edit** the already stored data. This is done with the help of **RecordID** field, which is a unique ID given to all of the records while creating. Based on the ID, the user is prompted for the field to be changed. After receiving the input, the respective field is changed as per the requirement. This way, the manager has the freedom to introduce a change based on the requirement.

'getRecordCount()' is a method that allows the manager to retrieve the count of number of records that are stored in the data structure. This is done with the help of UDP implementation ie, the request is in terms of socket-communication. The request is sent as a packet (Datagram) and the sender waits for the result to be sent back, and the record count of all three servers are returned back, which is explained more in detail while explaining synchronization.

'transferRecord()' is another method that can be executed by the manager. This method allows the manager to transfer a record from one server to another. This record maybe a student or a teacher record. This process is also performed synchronously in order to avoid errors. Also, the record that is being transferred is deleted from the original server to avoid duplicates in the hashmap. The transfer record functionality is achieved using UDP communication by using a Transfer record packet between the three udp servers. The udp server starts a new thread as soon as it receives the transfer record functionality and serves the request in a separate thread.

DIFFICULT PART OR CHALLENGES FACED:

Web Service Architecture: Implementing the Web service architecture and understanding the basic functionality of wsdl for all three server location was time consuming.

Maintaining Concurrency: Synchronizing the right functions to regulate the concurrent access.
Altering existing code: Interface implementation for web services and various annotations was difficult to understand.

UDP Communication:

The UDP functionality is made use for 2 main functionalities in the DCMS. First being the **get RecordCount()** method. Here the communication is between the 3 servers. The method returns the count of all records from all 3 servers. Any server upon receiving the request for getRecordCount(), the respective server gets the count of it's own data stored in the hashmap and then calls for UDP messaging for acquiring the count of other servers. Once, all of the count is made, the result is returned to the original callee server.
The other functionality that uses UDP is **transferRecord().** For this purpose, the required recordID is first fetched from the user. After doing so, the managerID, RecID and the location is transferred to the respective server where the server adds the record to the hashmap while deleting the record from the previous data- structure.

Data Structures used:

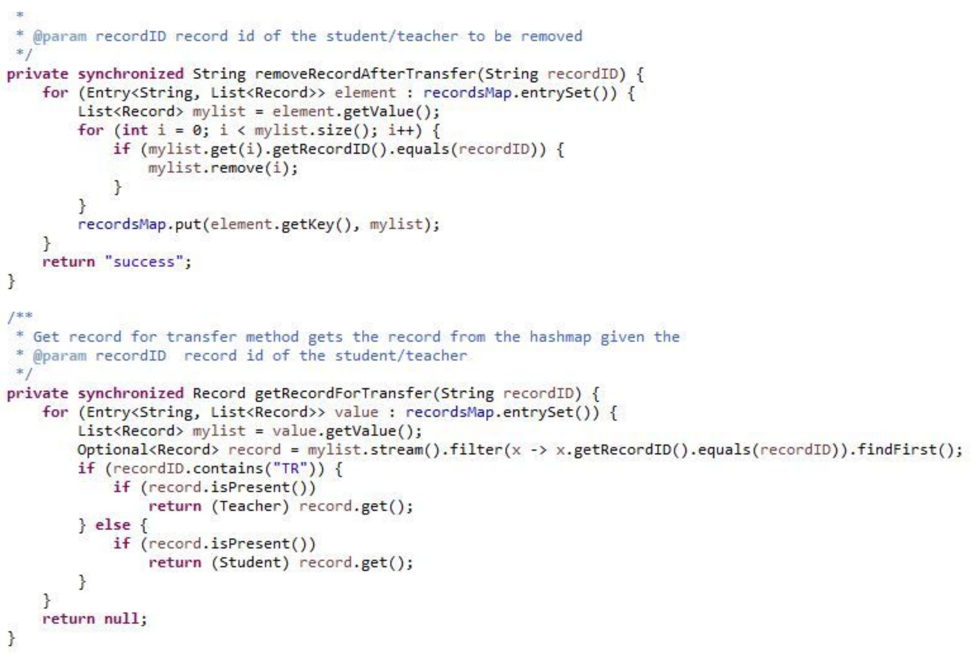
The teacher and student records are modelled as a Class that extends the base record class. The base record class holds attributes common to teacher and student. The records are stored in hashmap with the first letter of student/teacher's last name as key and the list of student & teacher records as value.
The Dcms interface is the file that exposes the methods of the server to the client, only the methods mentioned in the interface file can be accessed by the client.

Concurrency Implementation:

Concurrency factor plays a vital role when it comes to a distributed system with large amounts of records. This is because, many clients (managers in our case) can access the data concurrently which may lead to issues if not handled properly. Especially with many managers accessing at the same time from same/different locations, extra care must be put into concurrency implementation.

Multiple methods which are accessed from the client in the server implementation access common methods, which should be synchronized for concurrent access by multiple clients.

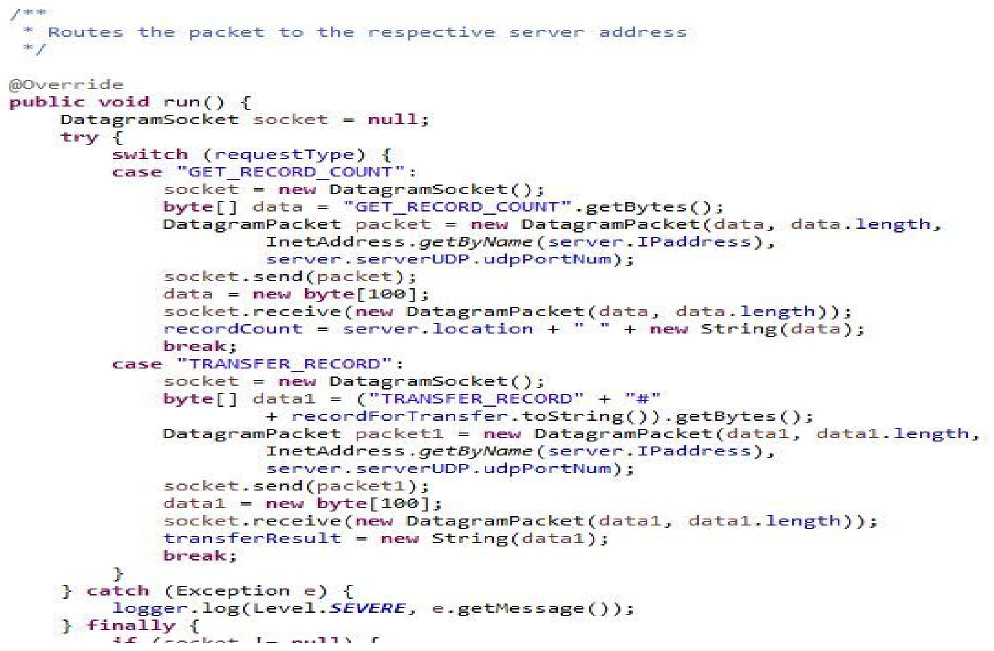
The DCMS implementation needs to be synchronized as multiple clients communicate with a server in the same location. We have implemented synchronization with the synchronized functionality that's available in JAVA. The server methods are all synchronized and the methods that are used to communicate from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



```java
 *
 * @param recordID record id of the student/teacher to be removed
 */
private synchronized String removeRecordAfterTransfer(String recordID) {
    for (Entry<String, List<Record>> element : recordsMap.entrySet()) {
        List<Record> mylist = element.getValue();
        for (int i = 0; i < mylist.size(); i++) {
            if (mylist.get(i).getRecordID().equals(recordID)) {
                mylist.remove(i);
            }
        }
        recordsMap.put(element.getKey(), mylist);
    }
    return "success";
}

/**
 * Get record for transfer method gets the record from the hashmap given the
 * @param recordID  record id of the student/teacher
 */
private synchronized Record getRecordForTransfer(String recordID) {
    for (Entry<String, List<Record>> value : recordsMap.entrySet()) {
        List<Record> mylist = value.getValue();
        Optional<Record> record = mylist.stream().filter(x -> x.getRecordID().equals(recordID)).findFirst();
        if (recordID.contains("TR")) {
            if (record.isPresent())
                return (Teacher) record.get();
        } else {
            if (record.isPresent())
                return (Student) record.get();
        }
    }
    return null;
}
```

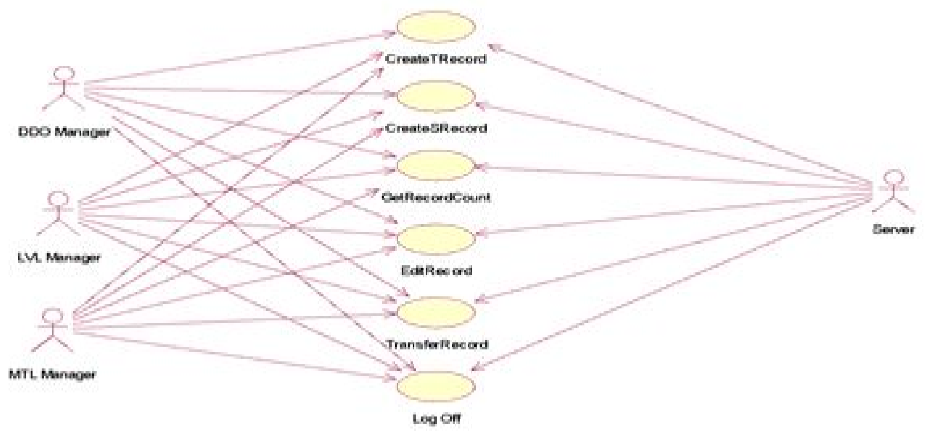
Synchronization among web services: Now, since each server location is implemented as a separate webservice, synchronizing multiple clients' access to different web services is necessary. In order to achieve it, **Java's synchronized feature** is used to specify which methods would be accessed concurrently, and moreover, the major functionalities like getRecordCount and transferRecord which are served as UDP requests, need to be made sure that same variables/methods are not accessed at the same time by multiple clients, for which every request to getrecordcount/transferrecord packet is served as a **separate thread** by UDPRequestProvider and UDPRequestServer classes, which allows faster and more reliable communication among web services and once each request is processed, the thread immediately dies. And moreover these two functionalities require synchronization among the three services, as the requests are handled not only between client-server, but also between servers.



```java
/**
 * Routes the packet to the respective server address
 */

@Override
public void run() {
    DatagramSocket socket = null;
    try {
        switch (requestType) {
        case "GET_RECORD_COUNT":
            socket = new DatagramSocket();
            byte[] data = "GET_RECORD_COUNT".getBytes();
            DatagramPacket packet = new DatagramPacket(data, data.length,
                    InetAddress.getByName(server.IPaddress),
                    server.serverUDP.udpPortNum);
            socket.send(packet);
            data = new byte[100];
            socket.receive(new DatagramPacket(data, data.length));
            recordCount = server.location + " " + new String(data);
            break;
        case "TRANSFER_RECORD":
            socket = new DatagramSocket();
            byte[] data1 = ("TRANSFER_RECORD" + "#"
                    + recordForTransfer.toString()).getBytes();
            DatagramPacket packet1 = new DatagramPacket(data1, data1.length,
                    InetAddress.getByName(server.IPaddress),
                    server.serverUDP.udpPortNum);
            socket.send(packet1);
            data1 = new byte[100];
            socket.receive(new DatagramPacket(data1, data1.length));
            transferResult = new String(data1);
            break;
        }
    } catch (Exception e) {
        logger.log(Level.SEVERE, e.getMessage());
    } finally {
        if (socket != null) {
```

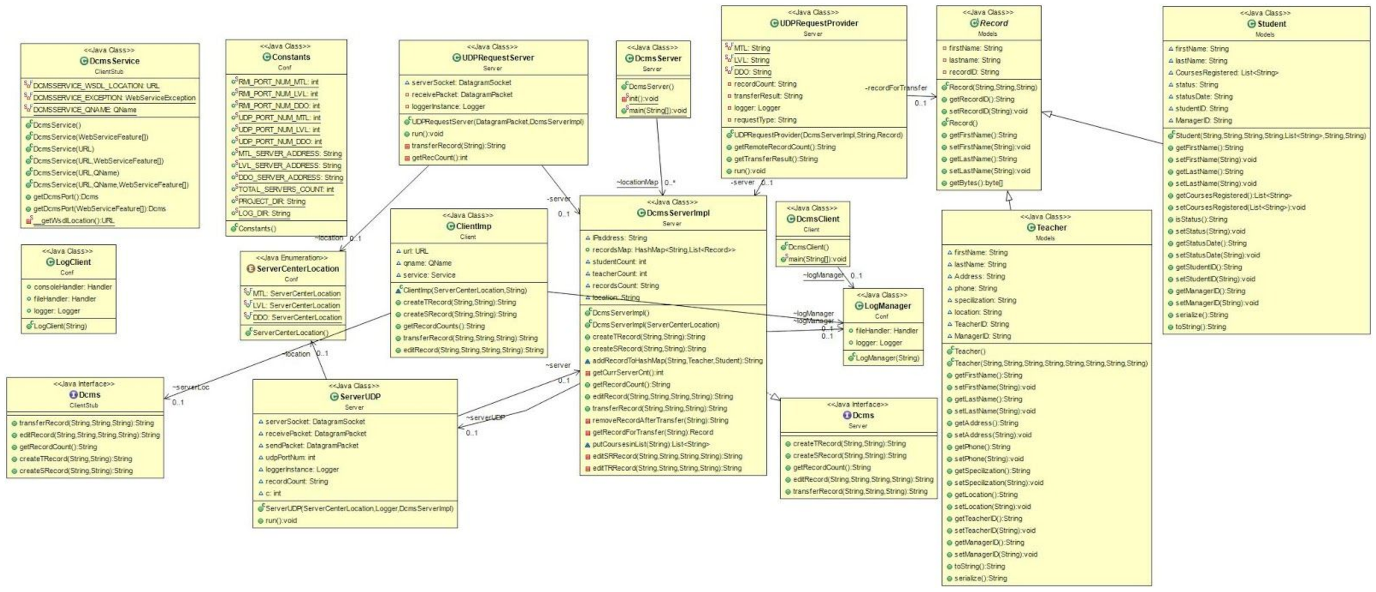
USE-CASE DIAGRAM



Description of UML Notation :

Use case diagrams are usually referred to as behavior diagrams used to describe a set of actions (use cases in the server side, such as the 5 major functionalities) that the DCMS system or systems should or can perform in collaboration with one or more external users of the system (client managers).

CLASS DIAGRAM:



Description of UML Notation :

A class is a classifier which describes a set of objects that share the same

- features
- constraints
- semantics (meaning).

A class is shown as a solid-outline rectangle containing the class name, and optionally with compartments separated by horizontal lines containing features or other members of the classifier.

The DCMS web service implementation consists of the following major classes :

Client - ClientImp and DcmsClient

Client Stub - Dcms and DcmsService

Configuration - Log

Models - Student, Teacher and Record

Server - DcmsServer, DcmsServerImpl, ServerUDP, UDPRequestProvider and UDPRequestServer.

TEST SCENARIOS:

The test scenarios and the various test input and output files can be found in the project in the path : **eclipse_workspace\Dcms_webservice\Test scenarios**

CONCURRENCY & SYNCHRONIZATION TESTING:

SINGLE SERVER - MULTIPLE CLIENTS TEST CASES:

Server was configured in one of the university lab machines with IP 132.205.4.65 ,the three web services were hosted in this machine. Clients were configured in the university lab machines 132.205.4.66 and 132.205.4.63, the web service client code was started.

1. Simultaneously requests for Create teacher record were issued from multiple clients - Server responded successfully.

2. Get record count and transfer record concurrently from multiple clients to the same server.

3. Edit the same record and the same field concurrently from multiple clients.

4. Transferring the same record from multiple managers to a different server at the same time.

Create 1 record, Get record count, Client 1 - transfer record from MTL -> LVL, record transferred , Client 2 - transfer record from MTL ->DDO , record not found message displayed.

MULTIPLE SERVERS - MULTIPLE CLIENTS TEST CASES:

Servers were configured in two of the lab machines with IP 132.205.4.62(LVL&DDO) and IP 132.205.4.65(MTL). Clients were configured in two machines with IP 132.205.4.66 and 132.205.4.63, the web service clients code was started.

Multiple clients should get record count from multiple servers running in different location.

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, or else if credentials are invalid, request again.	Yes, logged in, requests if credentials are invalid.
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.
T7	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)
T8	Transfer record to itself	The server should deny the process.	System made sure that the transfer did not happen.
T9	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.

T10	Multiple clients request to same server.	Serve the request to the appropriate client.	The server communicated and returned the result to the appropriate client.
T11	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 returned the correct result.

References:

1. https://www.techopedia.com/definition/25301/web-service
2. https://www.ibm.com/support/knowledgecenter/en/SSGMCP_5.3.0/com.ibm.cics.ts.webservices.doc/concepts/dfhws_definition.html
3. https://johnwsaunders3.wordpress.com/2008/09/23/basics-how-web-services-work/
4. https://docs.oracle.com/javaee/6/tutorial/doc/bnayn.html
5. https://www.uml-diagrams.org/class-reference.html