Department of Computer Science and Software Engineering

COMP 6231 - SUMMER 2018

DISTRIBUTED SYSTEM DESIGN

ASSIGNMENT 3

Distributed Class Management System (DCMS) using Web Service Implementation

Submitted by : Team #2

Design Architecture:

- **CenterServerImpl:**
  This class implements the interface mentioned above. There are 3 center servers : Montreal, Laval and DDO. Each has their own hashmap to store the student and teacher record. Initially there are few records for student and teacher. Hashmap maintains key and value pairs.
    - Here we have used Hashmap<String, ArrayList<Record>> where String as a key will contain alphabet between A to Z. Arraylist stores the list of students and teachers records according the first character of the last name. e.g. all the records whose last name starts with “A” will be stored in arraylist “a” and then put into hashmap with key “A”.

- **Server:**
  This is the main class for the server invocation. startServer() method will create 3 server instances of running on 8080 port , each instances, will create the naming services which allows clients to find objects based on names.

- **ClientManager:**
  It is a client which invokes the center’s server system to test the correct operation of the DCMS invoking Server.

- **LoggerFactory:**
  It to generate the log of the activities performed during the execution of the program.

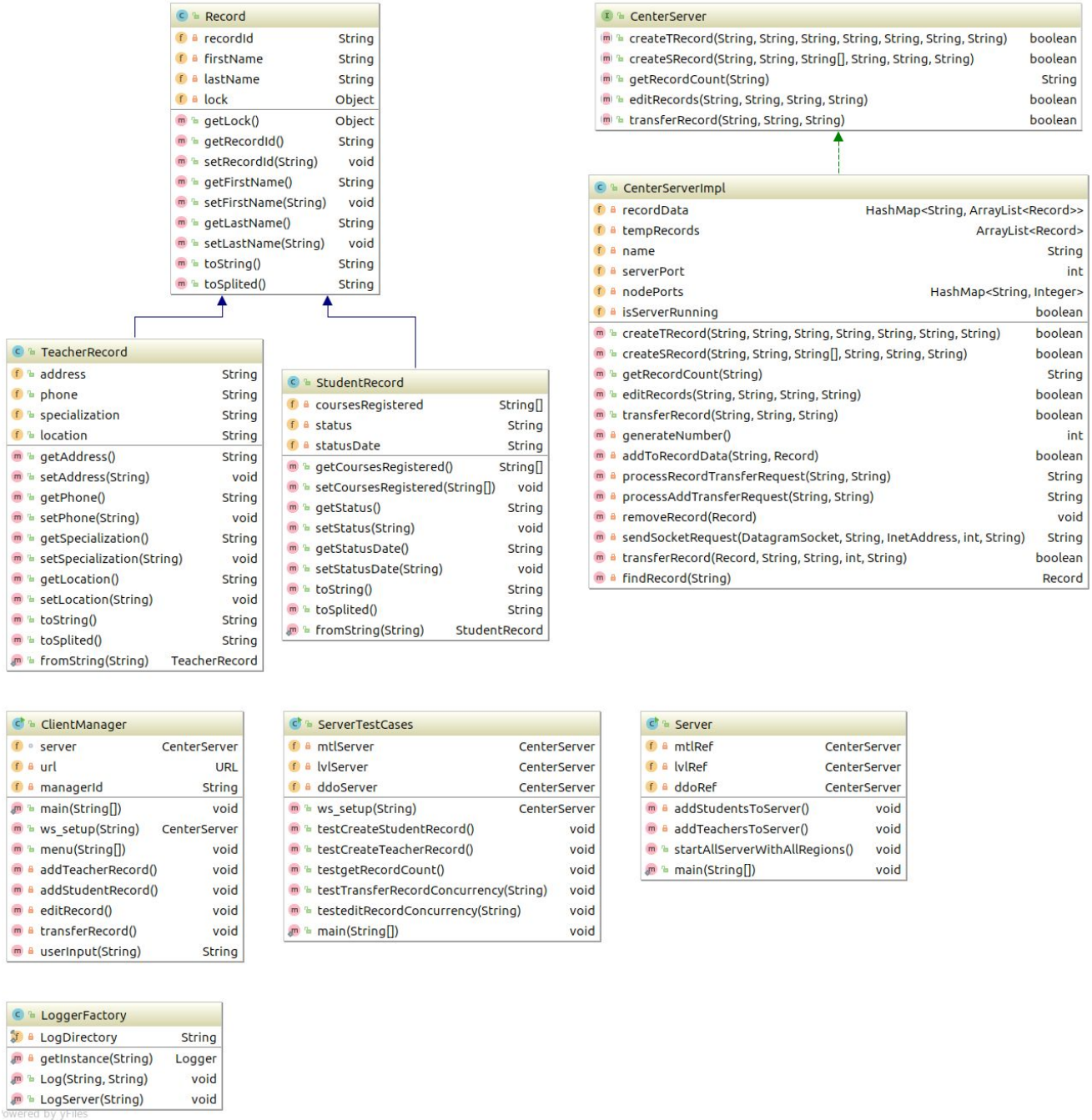
- **Record:**
  It contains the properties: recordId, firstName and lastName.

- **TeacherRecord:**
  It extends the Record class and contains the properties like address, phone, specialization and location.

- **StudentRecord:**
  It extends the Record class and contains the properties like status, coursesRegistered and statusDate.

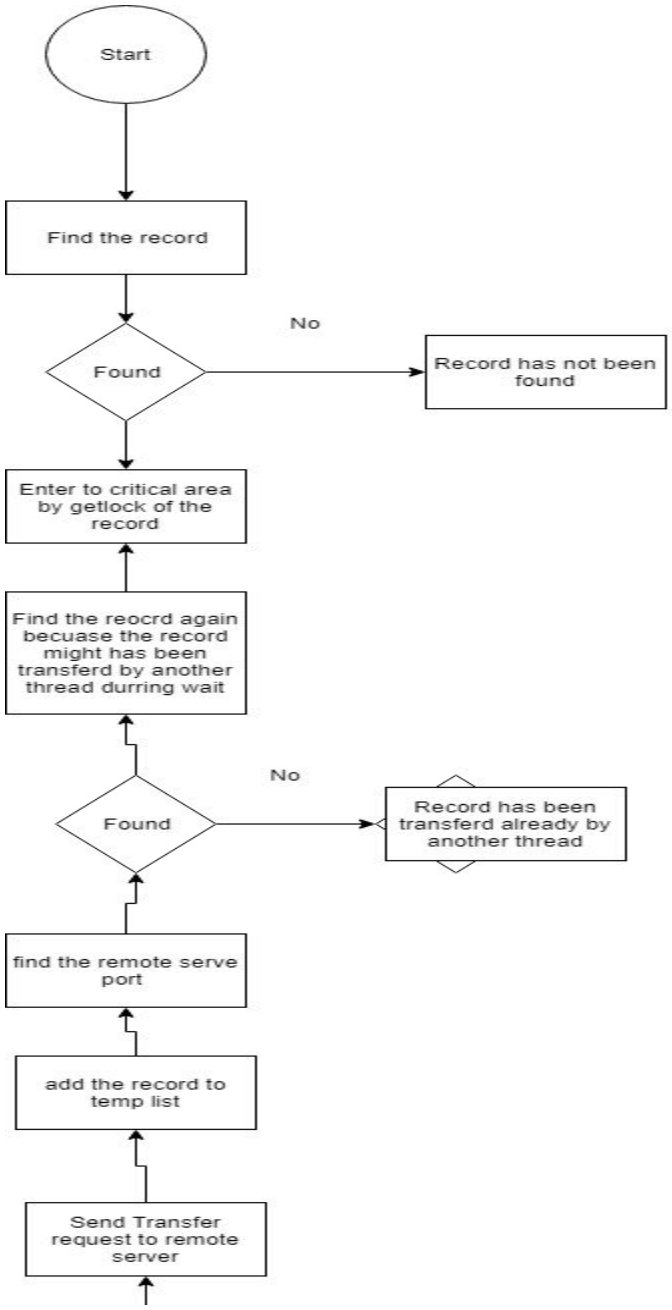
Class Diagram:

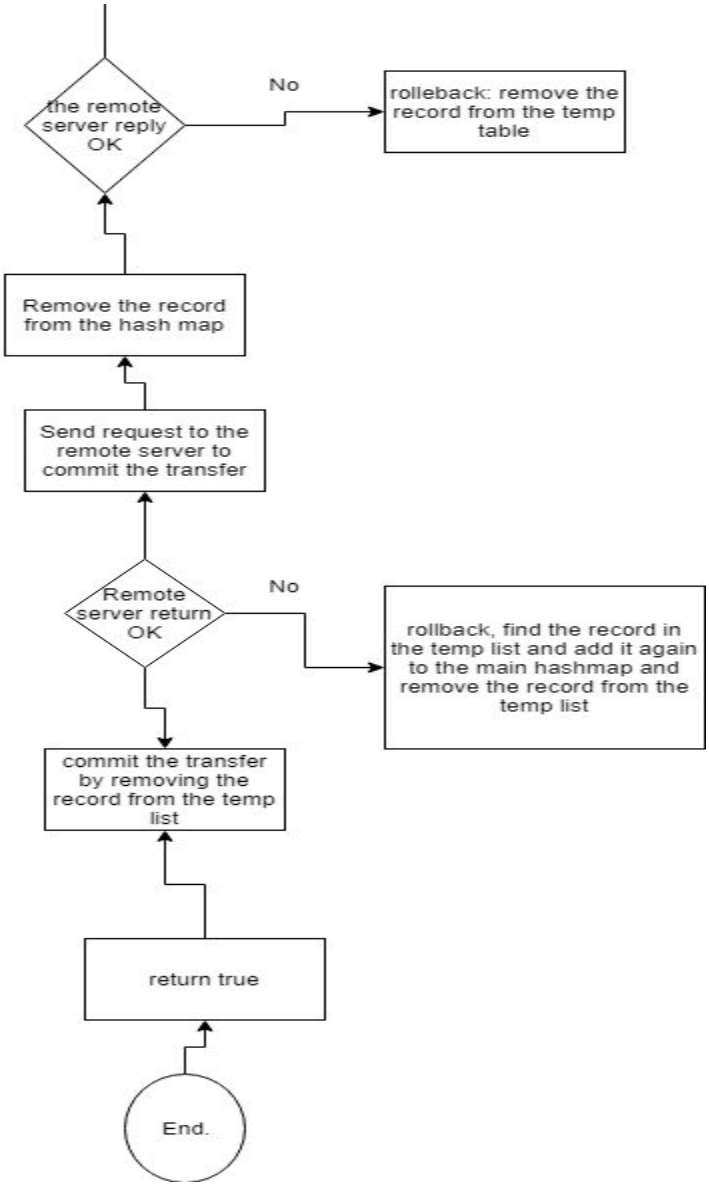
Following class diagram best describes our architecture that we have used:



Techniques Used:

- **Multithreading:**
  We are using Threads for parallel request to get the record counts from servers. One server will create appropriate threads and sends the request for getting record counts to remaining servers and will wait for their response.
  For the transfer record we have used threads and udp socket to communicate between servers. The below flow chart shows the workflow in the server side

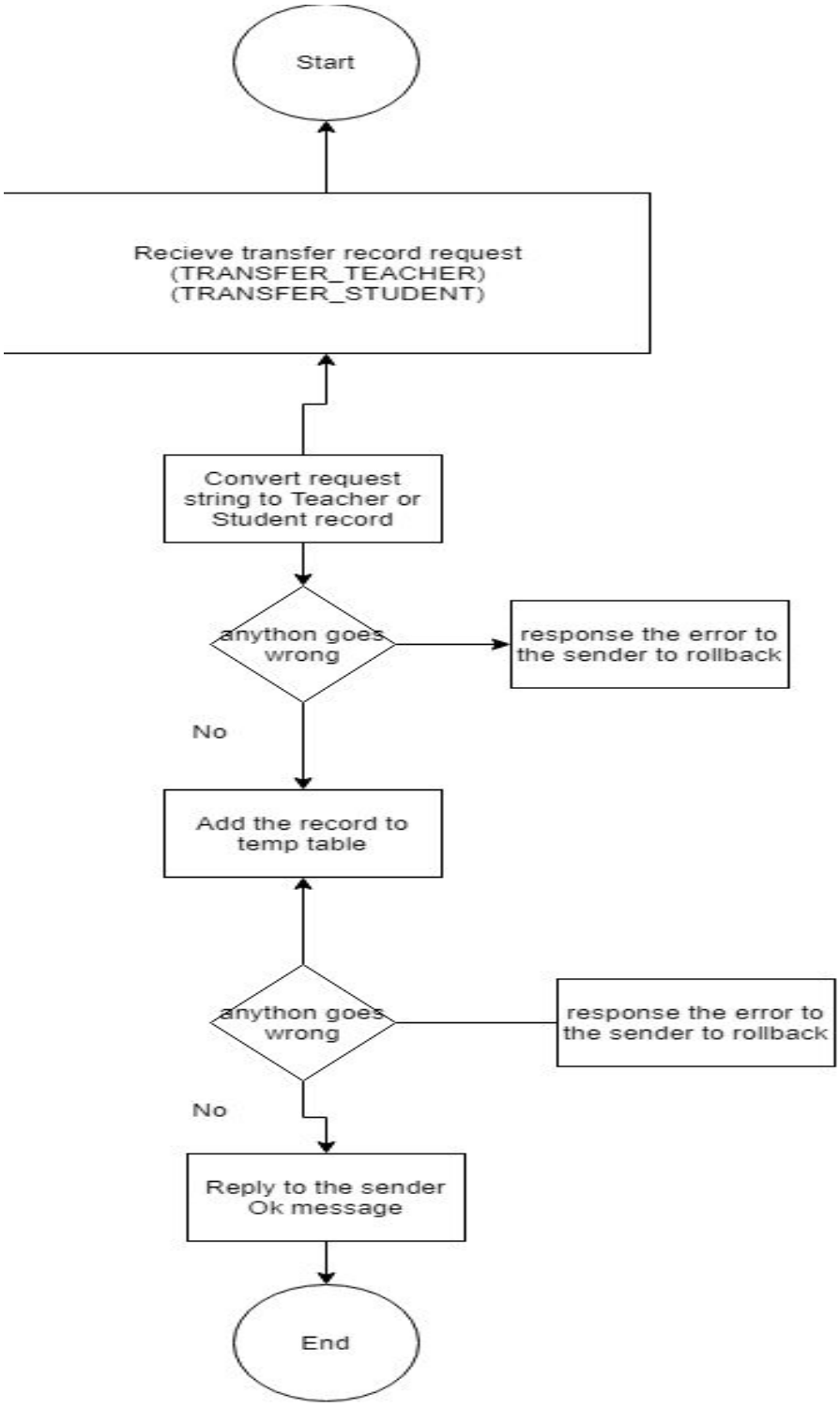


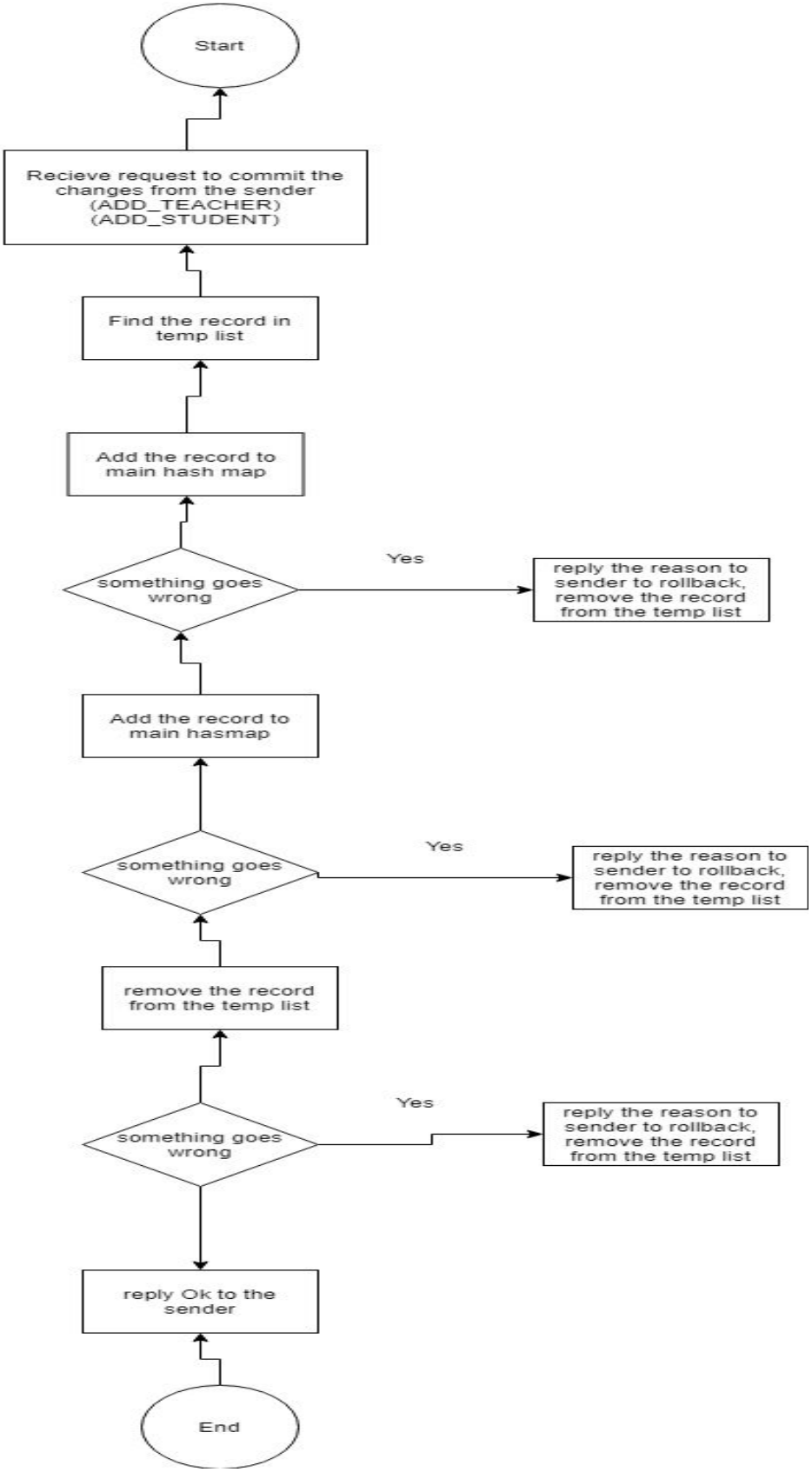


We are using the java synchronized to lock access a record for edit or transfer. We are also provided the transactional operation to make sure we will not lose a record. In fact at very first step we just add the record to a temp list on the server side and then a request to the remote server, the remote server does the same and reply the result to the server. If so far everything goes well the server will remove the record from the main hashmap and will send a request to the remote server to complete the transfer and add the record from the temp list to main hash map and the end the remote server will send OK message to the server and the server will

remove the record from the hash map and temp list, if any thing goes wrong in any step the both server and remote server can roll back the transaction, because the have a copy of the record on the their own temp list.

The following flow chart shows the workflow on the remote server




Start
Recieve request to commit the changes from the sender (ADD_TEACHER) (ADD_STUDENT)
Find the record in temp list
Add the record to main hash map
something goes wrong
Yes
reply the reason to sender to rollback, remove the record from the temp list
Add the record to main hasmap
something goes wrong
Yes
reply the reason to sender to rollback, remove the record from the temp list
remove the record from the temp list
something goes wrong
Yes
reply the reason to sender to rollback, remove the record from the temp list
reply Ok to the sender
End

- **Socket Programming:**
  Here we are using UDP programming. It uses datagram packets that provide functions to make a packet to transmit, which includes array of bytes containing message, Length of message, Internet address, port number. Datagram sockets provides support to send receive UDP datagram packets. Inetaddress is used for server. UDP programming is required because server has to communicate with other servers bi-directionally.

- **Web Service:**
  "Web Services are software components described via WSDL which are capable of being accessed via standard network protocols such as SOAP over HTTP."

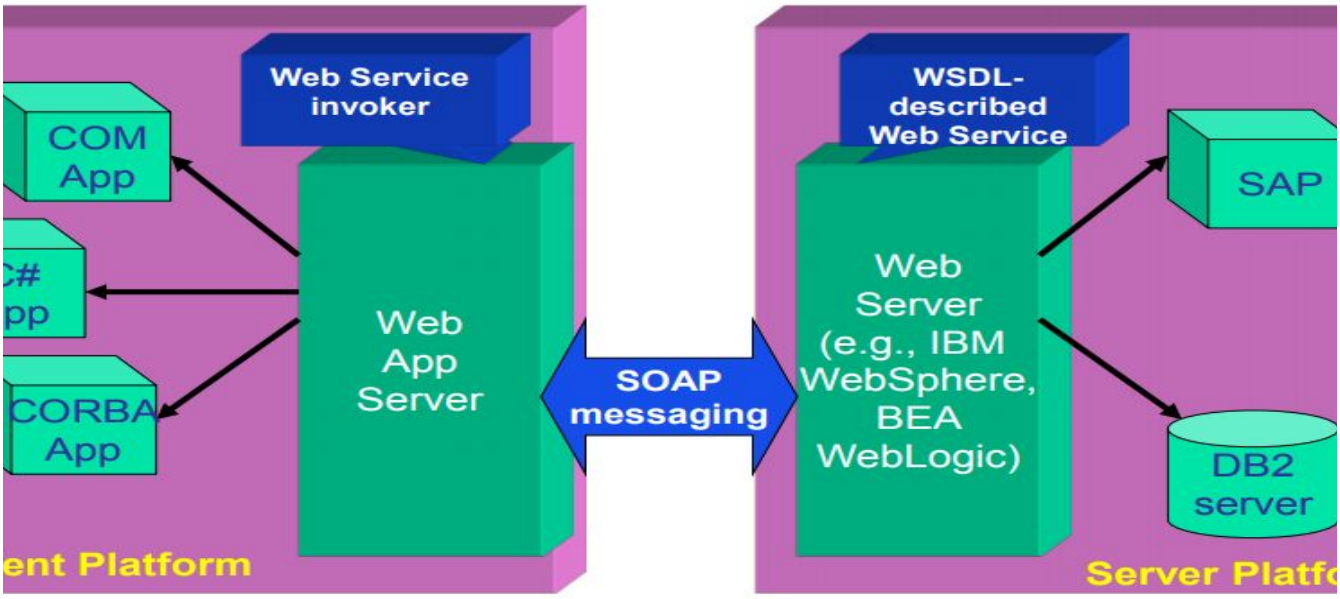


Image : Web Service Architecture

How it works ?

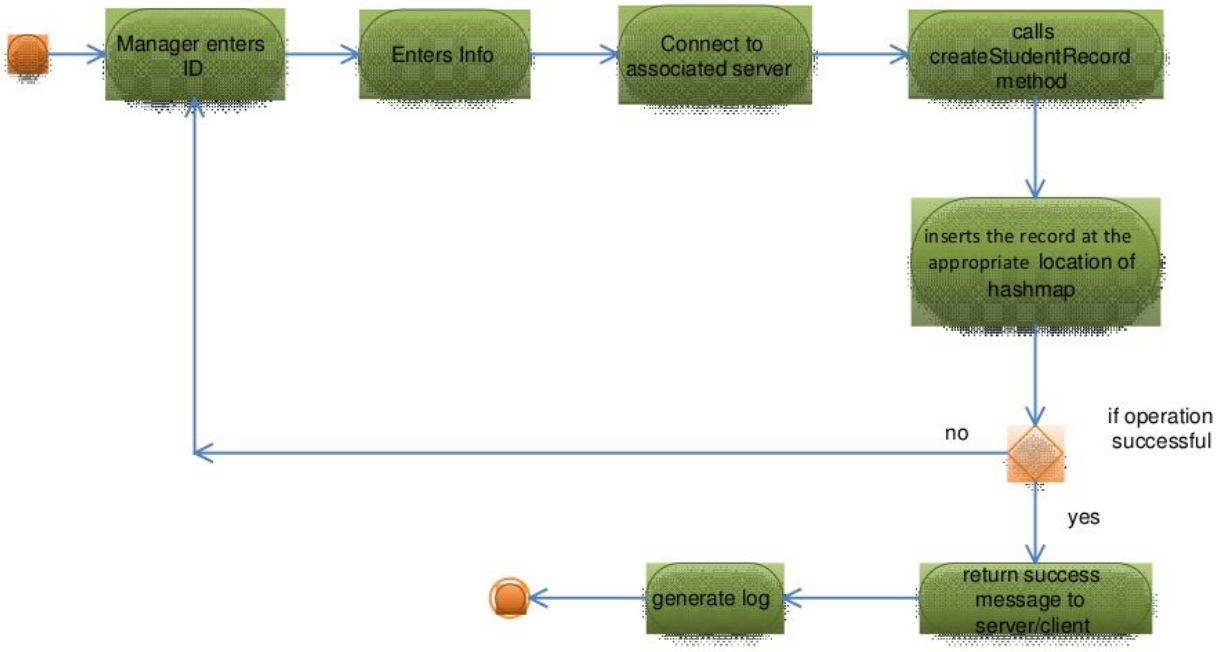
- First the client discovers the service.
- Typically, client then binds to the server. By setting up TCP connection to the discovered address. But binding not always needed.
- Next build the SOAP request: (Marshaling) .
- Fill in what service is needed, and the arguments. Send it to server side
- XML is the standard for encoding the data (but is very verbose and results in HUGE overheads)

- SOAP router routes the request to the appropriate server (assuming more than one available server) Can do load balancing here
- Server unpacks the request (Unmarshaling), handles it, computes result
- Result sent back in the reverse direction: from the server to the SOAP router back to the client.

Test Scenarios:

- **Create Student Record**

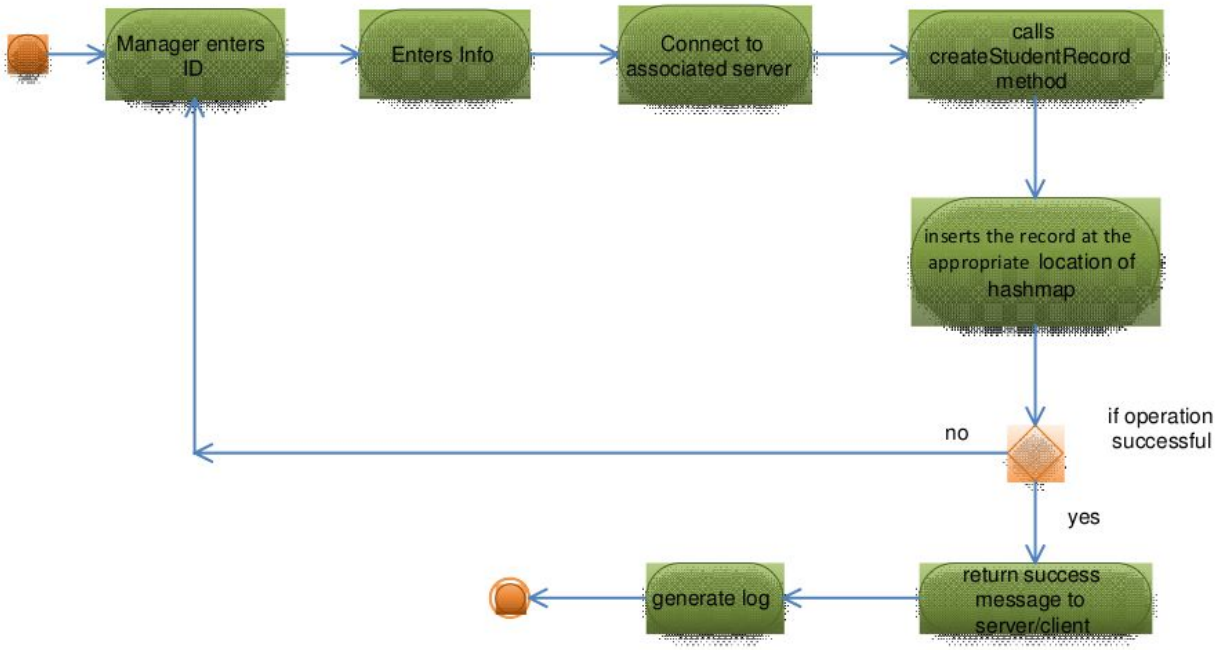
  To test the method that is responsible to create the new student record. When a manager invokes this method from his/her center through a client program called ClientManager, the server associated with this manager (determined by the unique managerID prefix) attempts to create a TeacherRecord with the information passed, assigns a unique RecordID and inserts the Record at the appropriate location in the HashMap. The server returns information to the manager whether the operation was successful or not and both the server and the client store this information in their logs.



- **Create Teacher Record**

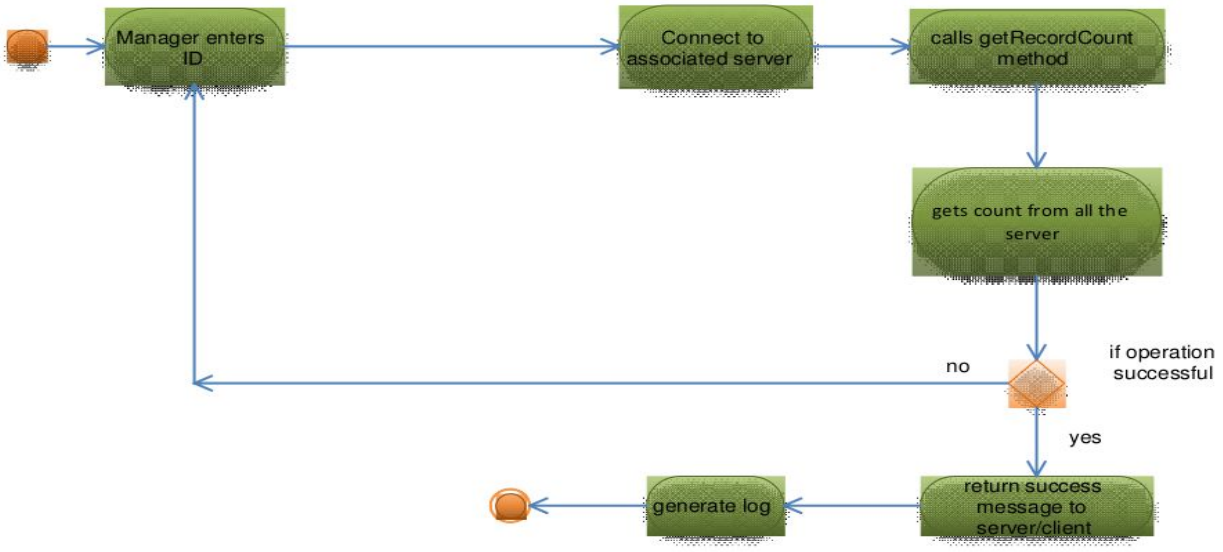
  To test the method that is responsible to create the new teacher record. When a manager invokes this method from a ClientManager, the server associated with this manager (determined by the unique managerID prefix) attempts to create a TeacherRecord with the information passed, assigns a unique RecordID and inserts the

Record at the appropriate location in the hash map. The server returns information to the manager whether the operation was successful or not and both the server and the client store this information in their logs.



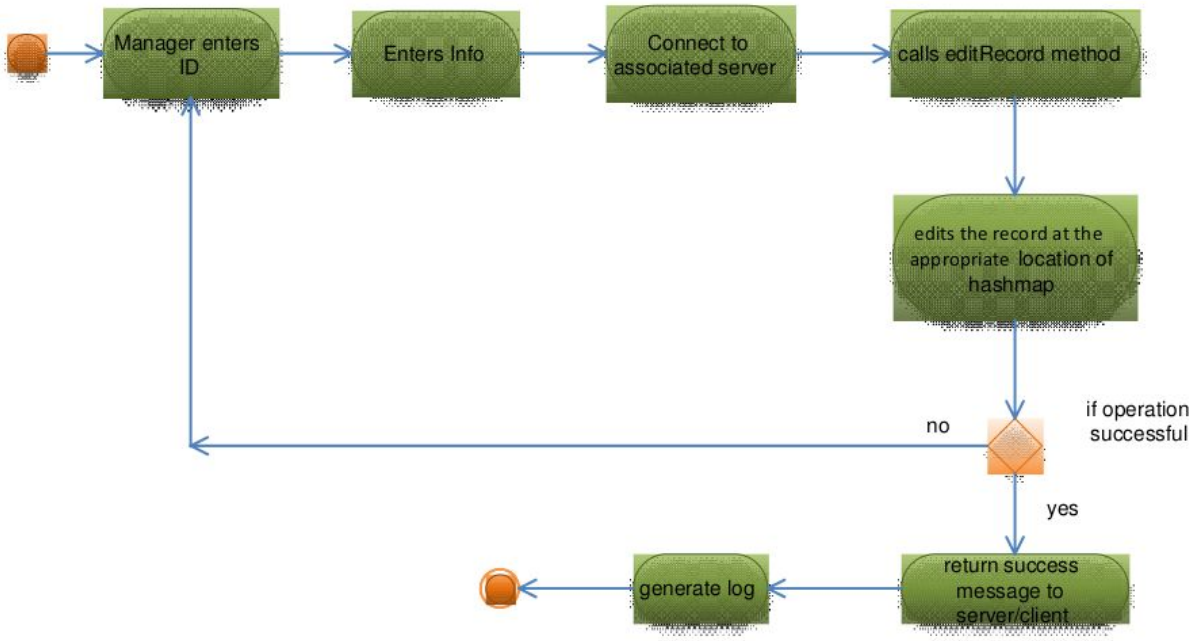
- **Get Record Counts**

  To test the method that is responsible to return the number of records stored on all servers. A manager invokes this method from his/her ClientManager and the server associated with that manager concurrently finds out the number of records (both TR and SR) in the other centers using UDP/IP sockets and returns the result to the manager. It only returns the record counts (a number) and not the records themselves. For example if MTL has 6 records, LVL has 7 and DDO had 8, it should return the following: MTL 6, LVL 7, DDO 8.



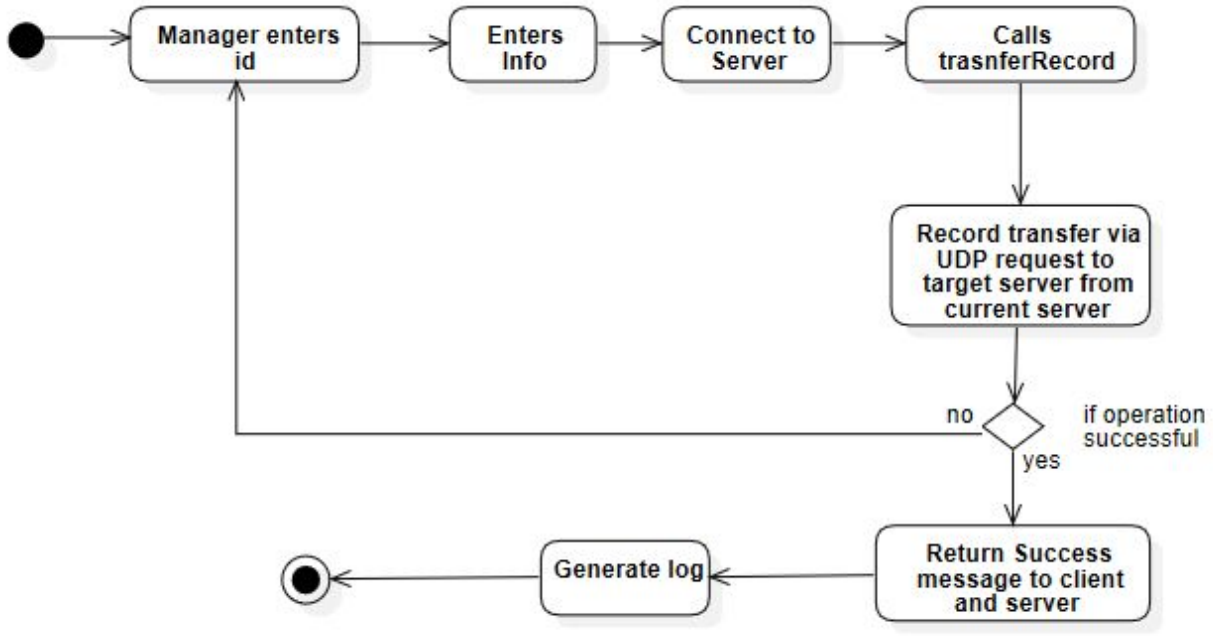
- **Edit Record**

  To test the method that is responsible to edit the record. When invoked by a manager, the server associated with this manager, (determined by the unique managerID) searches in the hash map to find the recordID and change the value of the field identified by "fieldname" to the newValue, if it is found. Upon success or failure it returns a message to the manager and the logs are updated with this information. If the new value of the fields such as location (Teacher), status (Student), does not match the type it is expecting, it is invalid. For example, if the found Record is a TeacherRecord and the field to change is location and newValue is other than mtl, lvl or ddo, the server shall return an error. The fields that should be allowed to change are address, phone and location (for TeacherRecord), and course registered, status and status date (for StudentRecord).



- **Transfer Record**

  When Transfer Record method invoked by a manager, the server associated with this manager, (determined by the unique managerID) searches in the hash map to find the recordID and transfer the record, if it is found. If Record is found then it locks the recordID and perform the transfer of record from hash map of current server to desired server via UDP. Upon success or failure it returns a message to the manager and the logs are updated with this information. It deletes the record from the current server's hash map.



One the most important functionality that we have added to our project is the logger function. By developing such functionality we could able to keep track of any state changes in the project and it is helpful to debug the project to find errors.

Secord, the most difficult part for us to maximize the concurrency. We need to use synchronized methods and lock mechanism to maximize the concurrency.

Test cases:

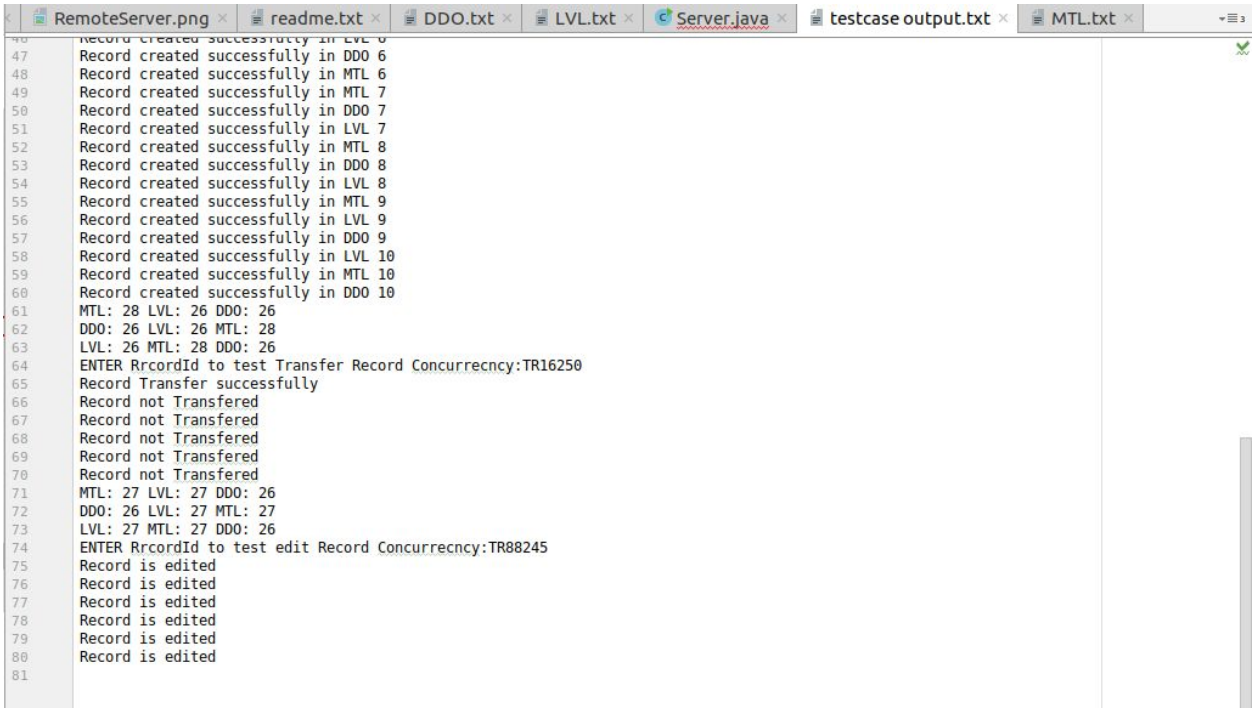
Multiple Manager Test case:

Java Thread Concepts are used to test this special case where we primarily focus upon the concurrency of the system that we have built for it. In a single instance we are creating multiple thread at same time and those threads perform actions on same server. Now each thread we can considered as a manager, so we have multiple managers logged into the system and all of them are trying to perform some actions on same server.

Different actions like getRecordCount and transferRecord and editRecord are tested.

Testlogs/ folder conatins all the log files related to the test cases. Each server has it's own log file.

ServerTestCase is the class file which have the test case.



For other test cases:

Please find the attached Excel file (“Testcases_Team#2.xlsx”) of test cases which includes the details of test cases, expected output, screenshots of Input and actual output.

References :

https://moodle.concordia.ca/moodle/pluginfile.php/3157537/mod_resource/content/2/Tutorial_6_Web_Services.pdf

https://moodle.concordia.ca/moodle/pluginfile.php/3157538/mod_resource/content/1/MyStoreServer.java

https://moodle.concordia.ca/moodle/pluginfile.php/3157491/mod_resource/content/4/5.Web%20Services.pdf