Department of Computer Science and Software Engineering

COMP 6231 - SUMMER 2016

DISTRIBUTED SYSTEM DESIGN

ASSIGNMENT 1

Issued: 17th May 2016 Due Date: 31st May 2016 by 5 pm

Note: *The assignments must be done individually or in a group of 2 (max) and submitted electronically.*

Distributed Staff Management System (DSMS) using Java RMI

In the assignments, you are going to implement a Staff Management System: a distributed system used by clinic managers to manage information regarding the doctors and nurses across different clinics.

Consider three clinic locations: Montreal (MTL), Laval (LVL) and Dollard-des-Ormeaux (DDO) for your implementation. The server for each clinic (called *ClinicServer*) must maintain a number of *Records*. There are two types of *Records*: *DoctorRecord* and *NurseRecord*. A *Record* can be identified by a unique *RecordID* starting with DR (for *DoctorRecord*) or NR (for *NurseRecord*) and ending with a 5 digits number (e.g. DR10000 for a *DoctorRecord* or NR10001 for a *NurseRecord*).

A *DoctorRecord* contains the following fields:

- First name
- Last name
- Address
- Phone
- Specialization (e.g. surgeon, orthopaedic, etc)
- Location (mtl, lvl, ddo)

A *NurseRecord* contains the following fields:

- First name
- Last name
- Designation (junior/senior)
- Status (active/terminated)
- Status Date (active or terminated date)

The *Records* are placed in several lists that are stored in a hash map according to the first letter of the last name indicated in the records. For example, all the *Records* with the last name starting with an "A" will belong to the same list and will be stored in a hash map (acting as the database) and the key will be "A". We do not distinguish between doctor records and nurse records when inserting them into the hash map (i.e. a list may contain both doctor and nurse records). Each server also maintains a log containing the history of all the operations that have been performed on that server. This should be an external text

file (one per server) and shall provide as much information as possible about what operations are performed, at what time and who performed the operation.

The users of the system are clinic managers. They can be identified by a unique *managerID*, which is constructed from the acronym of the clinic and a 4-digit number (e.g. MTL1111). Whenever a manager performs an operation, the system must identify the clinic that manager belongs to by looking at the *managerID* prefix and perform the operation on that server. The managers carry with them a log (text file) of the actions they performed on the system and the response from the system when available. For example, if you have 10 managers using your system, you should have a folder containing 10 logs.

The operations that can be performed are the following:

- *createDRecord* (*firstName*, *lastName, address, phone, specialization, location*) :

  When a manager invokes this method from his/her clinic through a client program called *ManagerClient*, the server associated with this manager (determined by the unique *managerID* prefix) attempts to create a *DoctorRecord* 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.

- *createNRecord* (*firstName*, *lastName*, *designation*, *status, statusDate*)

  When a manager invokes this method from a *ManagerClient*, the server associated with this manager (determined by the unique *managerID* prefix) attempts to create a *NurseRecord* 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.

- *getRecordCounts* (*recordType*)

  A manager invokes this method from his/her *ManagerClient* and the server associated with that manager concurrently finds out the number of records (both DR and NR) in the other clinics using UDP/IP sockets and returns the result to the manager. Please note that 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.

- *editRecord* (*recordID*, *fieldName, newValue*)

  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. For the new value for the fields such as location (doctor), designation and status (nurse), does not match the type it is expecting, it is invalid. For example, if the found *Record* is a *DoctorRecord* 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 DoctorRecord), and *designation, status and status date* (for NurseRecord).

Thus, this application has a number of *ClinicServers* (one per clinic) each implementing the above operations for that clinic and *ManagerClients* (one per clinic) invoking the manager's operations at the associated *ClinicServer* as necessary. When a *ClinicServer* is started, it registers its address and related/necessary information with a central repository. For each operation, the *ManagerClient* finds the required information about the associated *ClinicServer* from the central repository and invokes the corresponding operation.

In this assignment, you are going to develop this application using Java RMI. Specifically, do the following:

- Write the Java RMI interface definition for the *ClinicServer* with the specified operations.
- Implement the *ClinicServer*.
- Design and implement a *ManagerClient,* which invokes the clinic's server system to test the correct operation of the DSMS invoking multiple *ClinicServer* (each of the servers initially has a few records) and multiple managers.

You should design the *ClinicServer* maximizing concurrency. In other words, use proper synchronization that allows multiple police manager to perform operations for the same or different records at the same time.

Marking Scheme

[30%] *Design Documentation*: Describe the techniques you use and your architecture, including the data structures. Design proper and sufficient test scenarios and explain what you want to test. Describe the most important/difficult part in this assignment. You can use UML and text description, but limit the document to 10 pages. Submit the documentation and code electronically by the due date; print the documentation and bring it to your DEMO.

[70%] *DEMO in the Lab*: You have to register for a 5-10 minutes demo. Please come to the lab session and choose your preferred demo time in advance. You cannot demo without registering, so if you did not register before the demo week, you will lose 40% of the marks. Your demo should focus on the following.

  [50%] *The correctness of code:* Demo your designed test scenarios to illustrate the correctness of your design. If your test scenarios do not cover all possible issues, you will lose part of marks up to 40%.

  [20%] *Questions:* You need to answer some simple questions (like what we have discussed during lab tutorials) during the demo. They can be theoretical related directly to your implementation of the assignment.

Questions

If you are having difficulties understanding sections of this assignment, feel free to contact your Teaching Assistant. It is strongly recommended that you attend the tutorial sessions, as various aspects of the assignment will be covered.