Meeto: Collaboration and Social Networking

Distributed Systems 2014/2015

Deadline V1: October 24, 11:59pm

1 Objectives of the assignment

At the end of this practical project, you should have:

- Implemented a social network for managing meetings, following a client-server architecture.
- Mastered the usage of TCP/IP sockets for communication, using multi-threaded servers.
- Created a communication layer taking advantage of the Java RMI higher level API.
- Ensured the availability of the service, by implementing a failover solution.

2 Work description

Meetings are necessary when several people collaborate and work together. Students meet to coordinate their work on practical assignments; employees meet to discuss the status of projects; managers meet to make key decisions concerning their organisations; and so on. We need to prevent people from wasting time in useless meetings, because *time* is one of our most valuable resources.

The goal of this assignment is to build a distributed application to improve the effectiveness of meetings. The system shall be able to support the preparation of meetings, writing down the minutes, and following up on the status of action items resulting from a meeting.

3 Functional requirements

Every user that connects to your application should be given two options: register a new account, and login as an existing user. Once you've logged into the system, you may choose from several options:

- **Schedule a meeting.** The meeting leader creates a new meeting by setting a title, the desired outcome, date, time, and location for the meeting.

It is possible to immediately add other users that will be invited to attend the meeting and who will be able to access all the information concerning the meeting.

- **Set agendas democratically.** The meeting leader may initially add items to the meeting agenda. However, any users invited to the meeting may add, modify, and delete items from the agenda. The meeting agenda is therefore set democratically, with a last item "any other business" belonging to all agendas.
- **Discussion.** Each item on the agenda of a given meeting may be discussed within your application. In essence, each agenda item has a chat room where meeting participants can write down their comments and thoughts. Such messages should never be lost.
- **Finalizing a meeting.** Throughout the discussion, participants may add *action items* and *key decisions* to the result of the meeting. An action item is a task that is given to a user (not necessarily someone who attended the meeting). The user who becomes responsible for an action item should, after the meeting, be able to mark the action as done once it is finished. A key decision is the result of a discussion that took place during the meeting, and should be associated to the agenda item that was being discussed.
- **Check all meetings.** Any user may, at any time, use the application to check upcoming meetings, to read any information belonging to a past meeting, to accept/decline meeting invites, etc.
- **Manage user groups [only for teams with 3 students].** Instead of inviting users one at a time, groups of people working together should all be invited at once to attend a meeting. Having groups of users also allows one to easily select people to become responsible for action items at the end of each meeting. Therefore, it is necessary to create user groups and manage the members.

4 Non-functional requirements

The application should obey the following non-functional requirements:

- Clients should never be aware of network problems, except for long network failures.
- Data should be persistent. This means that crashes and network problems should not compromise the existing data, neither on the client nor on the server.
- Any operations requested by the user should not be lost. The application is not allowed to lose any operation even if it is not committed to the server.

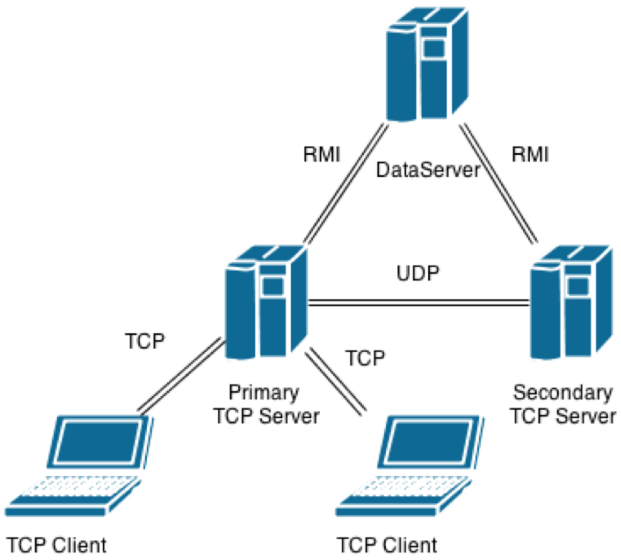

Fig. 1: Clients connect via TCP to a primary server and switch to the secondary in case of failure. Both TCP servers access persistent data stored in the RMI server.

- The messages exchanged in the discussion (of each agenda item) should be delivered to all clients in causal order.
- When a meeting agenda is being set, democratically, all operations should be transactional. If there is a network problem, one such operation should either be fully completed or fully reverted.
- Separation of concerns: the application's communication, business logic, and data should be separated and should be able to run on different machines (just requiring configuration).
- Any user interface is acceptable. Graphical user interfaces will not be rewarded when it comes to grading, so you should focus on the contents of the project, robustness, failover, and communication protocols.
- During the project defences, the demo has to be performed on 2 or 3 different computers (one per team member).

5 Software architecture

1. **Architecture.** First of all, you should design the architecture of your solution. You should have an overview of the components (server, client, database) and the details of how these connect to each-other (TCP, UDP,

RMI). Note that although you might run several components on the same machine during development, you should consider all components running in different machines after deployment. You should validate your architecture with the teaching staff before advancing to the next steps.

2. **Data structures.** You should stipulate how to represent your data, both in-memory and on disk. You should define the interfaces and classes that will hold Users, Meetings, Actions, Decisions, Comments, and Groups. You should also decide which persistence solution to use (binary files, text files, sqlite, mysql, postgresql, oracle, mongodb, etc.).

3. **TCP client-server.** You should then create two applications. The first is a client application for users to execute. That application should receive commands from the user, send them to the server and print the response. The other application should be a server that should be able to receive commands, modify the data-structures and reply to the client. You should focus on the availability and failover aspects of TCP communication, handling all possible problems in the networking stack.

   Your application should tolerate temporary failures in the TCP sockets between the client and the server. If the network or the server becomes unavailable for some time the client application will receive an exception. It should handle such exceptions and should try to open a new socket to the server. When there is no connectivity to the server, the application can warn the user that it is re-trying to connect.

4. **RMI server.** You will build an RMI Server that will wrap database/data operations to provide those operations over the network. It is a good idea to write the server interface (containing all methods) as early as possible, eventually in parallel with the data structures. You will then implement an RMI Client on your TCP Server to access those operations. The goal is to separate business logic from the communication logic. You should also focus on handling all possible errors in the best possible way. Note that the RMI Server may store data into files or an actual database to guarantee persistence.

   The TCP server should execute remote methods to handle the requests of each client (login, register, add new meeting, write message to an item discussion, and so on). Students should notice that an RMI remote object may receive multiple requests in parallel. Therefore, TCP servers and RMI server must deal with concurrent accesses. Clients need to receive notifications (for example, new discussion messages or new meeting invitations). To support this operation, callbacks are needed.

5. **UDP-based failover.** Machines also fail. Corrupted RAM, failing harddrives, burned BIOS, etc. In your deployment, your TCP-Server application should be executing in two different machines, being one of them the primary. If that machine stops providing the service, the secondary

machine should take over and they switch roles. If the failing machine recovers, it should then act as secondary.

Servers should ping themselves using UDP to ensure they are still running. When the backup server detects that the primary one is down (you can simulate this by shutting down the primary server using `Ctrl+C`) it should establish a connection to the RMI server (now as a primary TCP server) and resume TCP services to clients. This should be transparent for the clients. If a client detects that the connection is broken, it should try to reconnect and find the backup server instead.

6 Report

You should take time to write a report at the end of the project, considering the previous steps. You should write the report as documentation so a new developer may come in and understand how your solution was built, your technical decisions, and may introduce new components and modify or replace existing ones. You should include the following sections in your report:

- Introduction
- Internal architecture of the service (focus on interoperability)
- Data model of the application
- Protocol-specific communication (TCP, UDP, RMI)
- Exception handling in sockets and RMI
- Failover solution
- Causal order implementation
- Installation and configuration manual
- A table describing the tests made to the application (passed and failed)

7 What you will learn

With this project you will acquire practical competences in the following topics:

- Sockets programming in Java
- Exception handling in Java sockets
- Development of multi-threaded servers
- Java RMI
- Implementing a failover solution
- Implementing server-code with support for multiple protocols

8 Upcoming assignment

In the following assignment (V2), you will have to enhance your service by:

- Creating a web-based version (HTML/JSP/JavaBeans)
- Integrating the application with a third party web API.

9 Submitting your assignment

Deliver everything in one ZIP file. This file should include a `readme` file with *all the necessary information* to execute and test the assignment without the presence of the students. Assignments that do not contain this README with all the necessary instructions will not be evaluated. Assignments that do not execute correctly will also not be evaluated.

Inside the ZIP file, there should also be a PDF containing your report. Do follow the suggested structure, as all sections will be evaluated.

You should submit the ZIP on the inforestudante platform until October 24, at 11:59pm: `http://inforestudante.uc.pt`