Departamento de Engenharia Informática
Distributed Systems

2016/2017 - IVote

Report

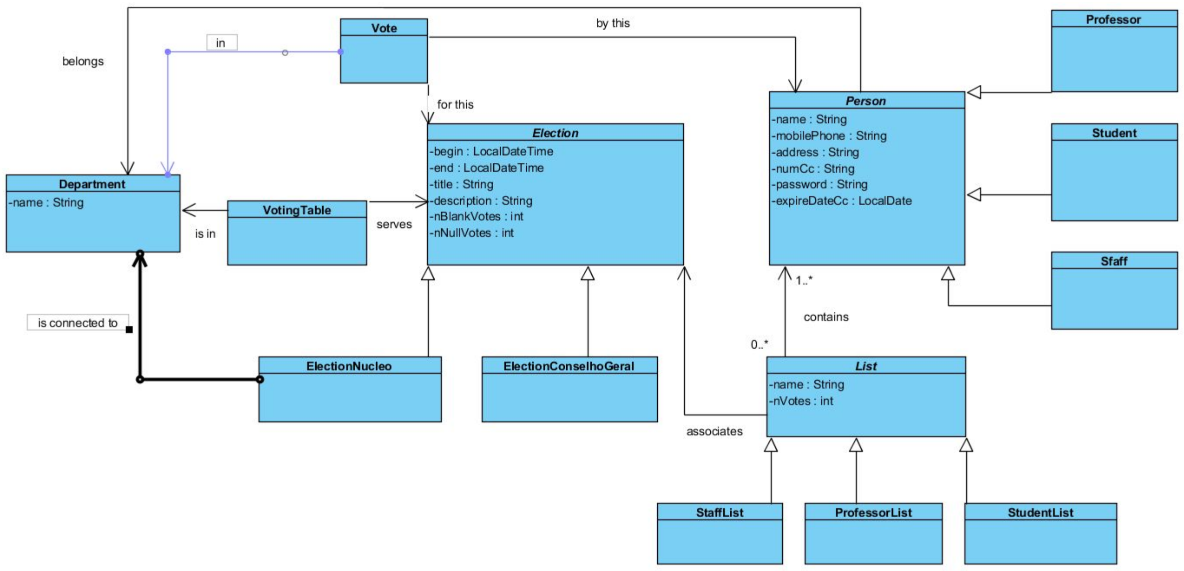
Joaquim Ramires Ferrer 2015260670
‹JJ›

Software Architecture

For this project we divided the source code in 5 different parts: the Admin Client, the RMI server, the TCP server, the TCP client and the general classes used by all of the other parts.

1. General classes architecture:

Simplified class diagram:



All classes serve mainly for storing information and provide little logic for the application except for:

- Subclasses of List only let add a Person to them if they meet the criterias. Only students are allowed on StudentList's and equivalently for the other subclasses.
- The Person class doesn't implement a getPassword(). Instead it implements a void checkPassword(String) method that verifies if the argument is equal to the password. This prevents the password from being accessed as is in the server. Ideally it would be hashed but that wasn't implemented.

2. RMI server

In the RMI server we have 3 classes: the main server, a database handler and a failover mechanism.

MainServer:

In this class, which implements a main method to be run in the primary and in the backup server, we have the methods that are to be called by the clients, both the admin client and the TCP server.

Most methods are normal calls to the database handler in order to offer the options to manage the data. Examples are create users, create voting tables, update elections, getting the elections that are associated with a voting table. It also offers methods to the tcp server that had to be done server side to guarantee the integrity of the application like the remote voting method that has as arguments every necessary field to check if the voting is done correctly, or the authenticate user to verify if a password given corresponds to a user identity card number (which is the identifier of the users) and its password.

In order to provide in real time the information about elections and voting tables to the admin, we save all admins that connect to the server in an array. Then, every time there’s a vote or a voting table connects, we push this information to every admin in that array.

To provide a reliable backup mechanism, every server has a boolean property regarding whether it’s the backupserver or not and a date where the server become the main server (the mechanism will be explained in the FailoverMechanism class).

When the main method is run we initialize a MainServer object and FailoverMechanism class. The way it recognizes whether it’s going to be the backup server or not by checking if there’s an object binded to the keyword “admin” in the registry.

It also starts a thread that pushes the necessary information to the subscribed admin consoles. It pushes notifications about on and off voting tables and the active elections number of votes. Subscribed admins are stored in an array.

FailoverMechanism:

This class extends the Thread class in order to provide a parallel execution to the server. To it it’s associated a MainServer object, the ports and addresses with which it will communicate with the other server, a buffer with size of 1024 bytes, a time between which the primary server should ping the backup server a time limit to which the backup server, if passed without communication from the primary, should take over the operations.

When an object of this class is created it starts by creating an UDP socket and connecting it to the other server. When run the method called depends on if it the MainServer associated with it is backup or not.

If backup, it will wait for a packet from the primary server. If it doesn't arrive in a certain time it will give order to it's associated server to take control. This means it will save the time of the event, rebind the "admin" keyword of the registry to itself, and start calling the primary server method on the this class.

If it's the primary server, it will send an UDP message to the backup server with a certain interval between them. This message will contain the date and time at which it became the primary server. Between each sent message it will listen for messages from the other server to look for the situation in which it's messages were delayed for some reasons and the backup took over. If it receives a message, it will contain the date and time the other started to work as primary server. If this time is after the time of the start of operations we know the other server is correctly bound to the registry so it assumes the backup server role.

Database:

This class represents a database handler. It organizes the information in files and is a last layer of protection to the integrity of the data. Each class in the general classes have a unique set of identifiers the database handlers respect. For this, we use a lock that is set in every method of the Database instance as it's performance is not important to this course.

The identifiers for each class are the following:

- Department : name
- Election : title
- List : election and name
- Person : numCc
- Vote : election and person
- VotingTable : department and election

Notice that the fact of having the same lock for every method and having identifiers for every class, makes every create and update operation of the kind "at least once". This simplifies the necessary logic in the application.

3. Admin client

In the admin client we have two classes. The AdminClient in which we do a first lookup to the rmi registry for the admin interface and a first subscription to the rmi server found. In the AdminInterface we have the main logic of the admin console.

AdminInterface

The admin Interface acts both as a server, to receive callbacks and as a client. It has all the options to manage the data in the server such as create and update elections, users, voting tables and departments with a lot of code to read input from the user.

Everytime it calls a method from the server it also checks for remote exceptions meaning the server is not up. It waits and calls the same operation again until a certain time has passed. Then it assumes the primary server is down and tries to rebind the rmi server. If it's not successful then it assumes both servers are down and shuts down.

It has 2 callable methods that update the active elections and their number of votes, to be seen after, and one to notify him about the voting tables that connect and disconnect.

4. TCP

In the TCP we have 4 classes TCPServer, TCPClient, ClientHandler, RegistrationProcess. This classes handle everything what is happening in voting point.

TCPClient:

The only 2 things that TCPClient does is:

1. Receiving information from Server, which are always very precise questions such as write password or chose a list
2. Sending answers, that voter write in the console.

TCPServer:

TCPServer implements main method and it is first program that has to be run in voting point(RMIServer must work already). At first the program asks for department name in which the election starts. The person who will be sitting at the voting table has to choose department from the list. After that three threads start to run:

1. RegistrationProcess thread, that provide all necessary registration option, which will be described later. As a constructor parameter it gets only departmentName.

2. Main thread, that listens and waits for new clients to appear.
3. Thread - simply runs remote method every few seconds to inform admin if it is on or off.

This is the moment to run terminals. Every single voting terminal is a TCPClient and needs it's ClientHandler. What is happening in the main thread is that when new client asks for connection, TCPServer starts new ClientHandler thread, (which gets also unique number). It pass also department name. From this moment when voter goes to voting terminal the communication protocol is described in ClientHandler and this class handle all answers from TCPClient. But details later. First voter has to registrate.

RegistrationProcess:

This class provides interface for person who is in charge of registering new voters. The procedure of registering and unlocking terminal looks as follows:

1. Program asks about voter numCc.
2. Person type it.
3. Program lists available elections for this department. To do so it runs remote method from RMI Server:

getVTableElections(departmentName);

4. Person chooses election.
5. Title of the election is set
6. Program has to check if the voter can take part in the election that chose, so it runs remote method on RMI server:

userCanVote(numCc, chosenElectionTitle, departmentName);

7. If the voter can vote, the program asks which terminal should be unlocked.
8. Person chose terminal and the registration process is over. Voter go to voting terminal. The rest of voting procedure is handled by ClientHandler class.

ClientHandler:

ClientHandler gets all necessary information to support vote method:
vote(*departmentName, numCc, electionTitle, listName, isBlank, isNull*);

- *departmentName* is transfered by constructor parameter.
- *numCc* and *electionTitle* are established in the moment when voter is linked to specific voting terminal.
- *listName, isBlank* and *isNull* are set during voting procedure inside ClientHandler class

TCP Protocol rules:

1. When voter comes to the terminal to which was assigned is asked for the password.
2. Voter has 3 chance to write correct password.
3. If password approved voter will see the election's lists.
4. Voter chose either one of the lists or blank vote. If voter write wrong answer then the vote is considered as corrupted/wrong/null.
5. *listName* is overwritten by name of chosen list or set as empty string. Respectively *isBlank, isNull* are set and it depends on voter choice.
6. The voting is over.

5. Run the program

The program was written using eclipse environment. To see how it works few things must be done in correct order:

1. Import 4 java projects to eclipse environment
    - VotingSystem
    - MainServer
    - ivote
    - AdminClient
2. Change Run Configuration of TCPClient and TCPServer class
    - right click on class in Project Explorer -> Run As -> Run Configuration...
    - go to Arguments lap
    - In Program Arguments write localhost
    - do that for both TCPClient and TCPServer classes
3. Start the rmi registry with the appropriate command in the bin directory of the ivote project.
4. Run RMI Server two times
5. Run TCPServer
    - answer first question to TCPServer console about department name.
6. Run TCPClient classes as much as it is needed
7. Everything is prepared to registrate voter and proceed voting
    - follow instruction in TCPServer Console
    - when you unlock particular terminal go to it's console and follow instruction.

6. TCP test

1. When the program run go to TCPServer
2. Choose 5 - DEI as a department name

3. write “1” to write person numCc, press enter
4. write “1” to set person numCc, press enter
5. write “1” to choose election, press enter
6. this election is not allowed for this person
7. write “0” to choose election, press enter, this election is available
8. this election is available
9. write “2” to chose terminal, go to correct terminal console
10. write password: benfica
11. chose list to submit vote.

7. Admin Test

After the start of the rmi registry and two servers.

1 - Create a nucleo election and check if it appears when printing the elections-Passed

2 - Create a voter and try to vote with him - Passed

3 - Create department and see if it appears when printing the departments - passed

4 - Create a list with wrong people and check if the creation fails - passed

5 - Create a good list - Passed

6 - Create a voting table and check if it’s now possible to connect a voting table in that department - Passed

7 - Turn on a voting table and check if it a notification appears in a connected admin - passed.

8 - Try to perform an operation on server after shutting it down and check if the operation is done after the time to do the rebind. - Passed

Task Distribution:

Joaquim Ferrer - RMI server and Admin Client
Jędrzej Adamczyk - TCP connection, proceed voting