RMI-based chat application: Papinho

Andon Tchechmedjiev, Jander Nascimiento

1 Presentation

We have developed here a graphical RMI-based chat application named papinho as we were instructed. The features implemented are the following : Sending of messages to a common public room, sending of private messages to any of the connected users on the public channel (except oneself), Changing one's name not only locally but on all the other clients as well, a full persistent history of the public channel on the server.

The architecture we have chosen for our application is a classical server centric one. Namely, all the clients are synchronized to a central state kept on the server. All messages be they public or private messages, will always transit through the server, where logging will occur.

2 Remote interfaces

We will first define the remote interfaces we used, as they constitute the basis of our application. There is one remote interface for the server application and one for the client application.

2.1 Chat server side

Each of the methods of the remote interface listed in table 1 represents the server part of one of the features above (except for the history).

Method	Summary
addClient	Add a client to the list of clients on the server side and notify the other clients; returns the list of users and the history to the client
removeClient	remove a client from the list of clients and notify the other clients
sendMessage	broadcast a message to all the clients (public chat), or to a specific client(private chat)
clientNameChange	Change a user's name and notify the other clients

Table 1: Server Side remote interface methods

2.2 Chat client side

Table 2 lists the remote methods for the server as well as a short description:

Method	Summary
addClient	Add a client to the list of clients on the client side
removeClient	remove a client from the list of clients on the client side
receiveMessage	Add a new message to the main view
receivePrivateMessage	Add a new message to the appropriate private chat window
changeClientName	Change the username in the list of clients on the client side

Table 2: Client Side remote interface methods

3 History and persistence

Here, we handle the persistence of the history, by serializing the *History* class directly to the disk, namely *bean serialization*. We also use the Proxy design pattern for the instantiation of the right DataSource.

4 Graphical User Interface

The Graphical User Interface(GUI) allows the user to interact with the application in a more intuitive and convivial way. Our graphical interface was implemented using the swing library as well a the Netbeans interface editor.

Figure 1 illustrates the main frame of our application, which contains: on the bottom left the text input field, in the bottom right, the Send button, on the top right side the user list and on the top left side the output text field (which supports colours for the usernames). The menu bar allow access to some useful function.

In the File menu can be found an *Options* menu item, which displays the dialogue that allows changing name, as well as the *Connect* menu item opening the dialogue which allows to connect to the server. Once the client is connected, the *Disconnect* menu item replaces *Connect*.

While there is no text in the input field, the send button is disabled, and the action of the Send button is supplemented by a press on the key Enter. As for the help menu it just contains a menu item that displays the about dialogue.

When a private chat is initiated, a private chat window pops up, in which it is possible to talk privately to whoever is on the other side of the line.

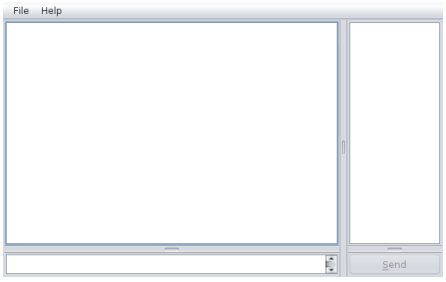

Figure 1: Main Window

5 Networking and security issues

Due to Java's default security policy, it is necessary to write a security policy file so as to allow clients to connect to our server remotely (by default only connections incoming from localhost are authorized). Additionally, we needed to authorize the use of shutdown hooks as we use them to perform cleanup operations when the server JVM exits. Our policy file is exhibited in listing 1. The policy file tu use is specified at runtime with -Djava.security.policy=./server.policy

Listing 1: Policy file

```
grant {
  permission java.security.AllPermission;
  permission java.net.SocketPermission "*:*", "accept
      ,listen,connect,resolve";
  permission java.lang.RuntimePermission "
      shutdownHooks";
};
```

We had a problem connecting clients from actual different hosts to the server, and as it turns out, the problem was that the server was bound to the wrong network interface. Indeed the UnicastRef returned to the clients pointed the ip address of this wrong interface, which in turn prevented the clients from connecting. In order to solve the problem, we had to manually specify the hostname to which to bind the server (bind in Socket terms not in RMI terms) with the runtime option -Djava.rmi.server.hostname=ipoftherightinterface.