Project

Daniel Hagimont - Daniel.Hagimont@irit.fr

USTH – Teaching Unit BC3.01

March 2024

Objectives

We want to implement a Java-based instant messaging (chat) service between users using distributed machines. It is assumed that all user machines have public IP addresses. In the initial phase, we are exclusively utilizing RMI (Remote Method Invocation) technology.

The application used by users is implemented by a class called ChatClient. A prototype of this class is provided (implementing a graphical interface). Internally, ChatClient implements the following static methods:

```
public class ChatClientImpl {

// called when the user (username) connects to the messaging service
public static void enter(String username) {...}

// called when the user (username) disconnects from the messaging service
public static void leave(String username) {...}

// called when the user wants to display the list of connected users
public static void who() {...}

// called when the user (username) has entered a message (text) for all users
public static void write(String username, String text) {...}

// called to display a message (text) sent by a user (username)
// in the graphical interface of the application
public static void print(String username, String text) {…}

}
```

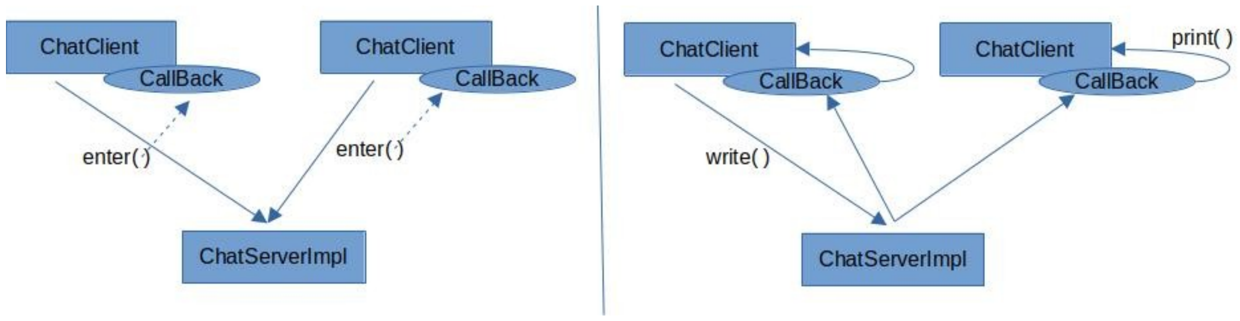
An RMI server object is used to implement communications between different instances of the application. This server object has an interface named ChatServer and is implemented by the class ChatServerImpl.

When ChatClient.enter() is called, the server object is invoked to register the new connected user within the server. A callback object must be passed as a parameter, allowing the server to call back to the client.

When ChatClient.leave() is called, the server object is invoked to unsubscribe the user.

When ChatClient.write() is called, the server object is invoked to broadcast the message. The server, which has registered callback objects for clients, can then call back to clients to display the message (using ChatClient.print()).

The following figure illustrates this scheme.



An interface for the server is provided below.

```
public interface ChatServer extends Remote {
      public void enter(String name, Callback cb) throws RemoteException;
      public void leave(String name) throws RemoteException;
      public String[] who() throws RemoteException;
      public void write(String name, String text) throws RemoteException;
}
```

First part

You must provide a Java RMI implementation for the instant messaging service. This implementation will include:

- the implementation of the ChatClient class
- the implementation of the ChatServer interface and the ChatServerImpl class
- the implementation of the callback interface and the callback class (CallBack/CallBackImpl)

Second part

In this second part, we will additionally use TCP sockets.

Now, we want the messaging system to allow broadcasting files (referred to as attachments) to connected users. The ChatClient application includes a new static method:

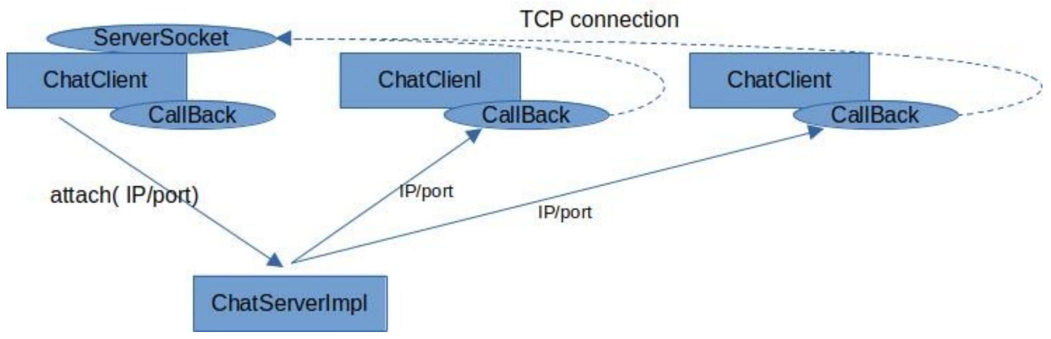
```
public class ChatClient {

      ….

      // called when the user (username) has selected
      // a file (filename) to broadcast to all users
      private static void attach(String username, String filename) {...}
}
```

The graphical interface must be modified to allow users to select an attached file and send it. This action triggers the invocation of the previously added static method (ChatClient.attach()).

When ChatClient.attach() is called, the server object is invoked to broadcast the attached file to all connected users. Since the file may be large, it is not transmitted to the server with RMI. The sending client of the attachment creates a ServerSocket to accept connection requests from recipient clients and sends the file over these connections. The RMI server call includes the

IP/port of the sending client as a parameter. The server can then call back to the recipient clients (using callbacks) as before, and communicate the sender's IP/port to them, allowing these clients to connect to the sender and retrieve the file content. The ServerSocket created by the sending client can be closed once all recipients have retrieved the attachment.

The following figure illustrates this scheme.



The client which broadcasts the attachment sends (via callbacks) the IP/port information for downloading the attached file. The attachment file is then downloaded and saved on all clients, except for the sending client.

To implement this feature, you need to modify the previous implementation. Clone (and keep a backup of) the initial implementation to implement the second one.

Expectations

You must implement the two versions of the application.
You must write a report (less than 3 pages in total) describing :

- your achievements (less than 1 page)
- a tutorial for using you tool (less than 1/2 page). Your tool should be easy to use (I should be able to test it on my laptop (locally) in less than one minute).
- add a signed letter saying the work that you returned to me is only from your own and that you did not copy anything from the Internet or from other students.

Project are done individually. You must sent to me an email with the source code of your contributions and the report. The dead-line is March 2024, the 23th (23:59 Hanoi time). If you return something more than 1 minute after the deadline, I will not evaluate your work and it will be a 0.