DISTRIBUTED SYSTEMS

Remote Methods Invocation

Master M1 MOSIG, Grenoble Universities

GRÄBNER Laurent
TCHOUGOURIAN Tigran

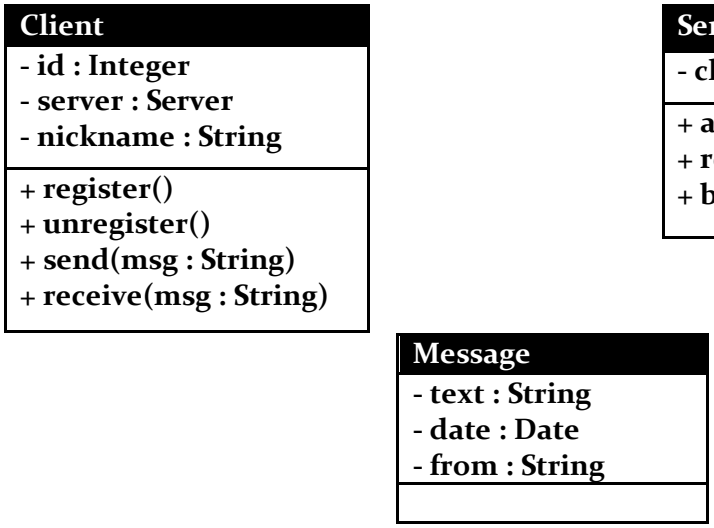
14/03/2011

Supervisor :
BOUCHENAK Sara

The assignment is about building an RMI-based chat system, where participants can dynamically join, leave, exchange messages.

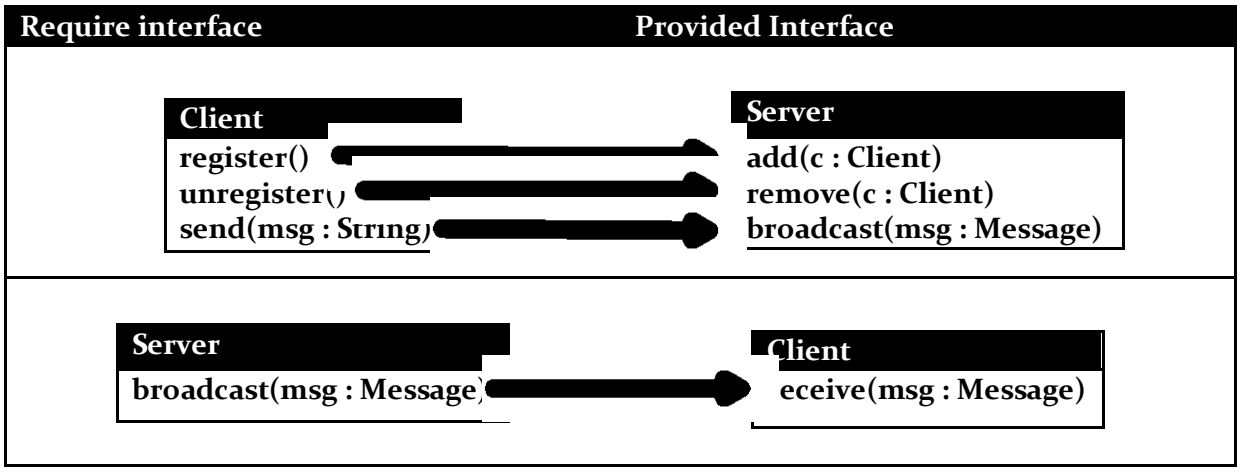
Table of contents

1. Objects of the application



Every client is represented by an object client; the server is unique in our design and stores all the clients. The Message carries the information from one client to the others.

2. Remote objects and regular Java object



- Client and Server are remote objects.
- Client and Message has to be serializable.
- Message is a regular Java object, because no methods are called on that object remotely.

3. Interfaces and classes of Remote objects

We define a naming convention. When an object has an interface and is remote, then it name's ends with the keyword "Remote" otherwise it ends with "Local".

3.1. Interfaces

3.1.1. ClientRemote

This interface is made to handle the remote method call `receive(msg : Message)`. The method allows the server to send a message to a specific client. In case of broadcasting, it should browse the list of connected client and send a message.

3.1.2. ServerRemote

The interface is made to handle multiple remote calls.

- add(c: Client)
  This function allows the server to add a new client to the chat.
- remove(c: Client)
  This function allows a registered client to be removed from the chat.
- broadcast(msg: Message)
  This function allows a server to send a message to all clients present in the chat.

3.2. Classes

3.2.1. Client

The regular Java object Client implements the interface ClientRemote.

3.2.2. Server

The regular Java object Server implements the interface ServerRemote.

4. Interfaces and classes of regular Java objects

4.1. Interfaces

4.1.1. ClientLocal

This interface is made to provide to the client methods to be able to use the chat.

- register()
  This function allows the client to be registered (connect) to the chat.
- unregister()
  This function allows the client to finish the chat session (disconnect).
- send(str : String)
  This function allows the client to send messaged on the chat to communicate.

4.1.2. Other

Message and Server do not need any local interfaces (regular classes). That because there is no local call on them.

4.2. Classes

Message and Server are never used locally, so there is no need to created interfaces for them.

4.2.1. Client

The regular Java object Client, implements the interface ClientLocal.

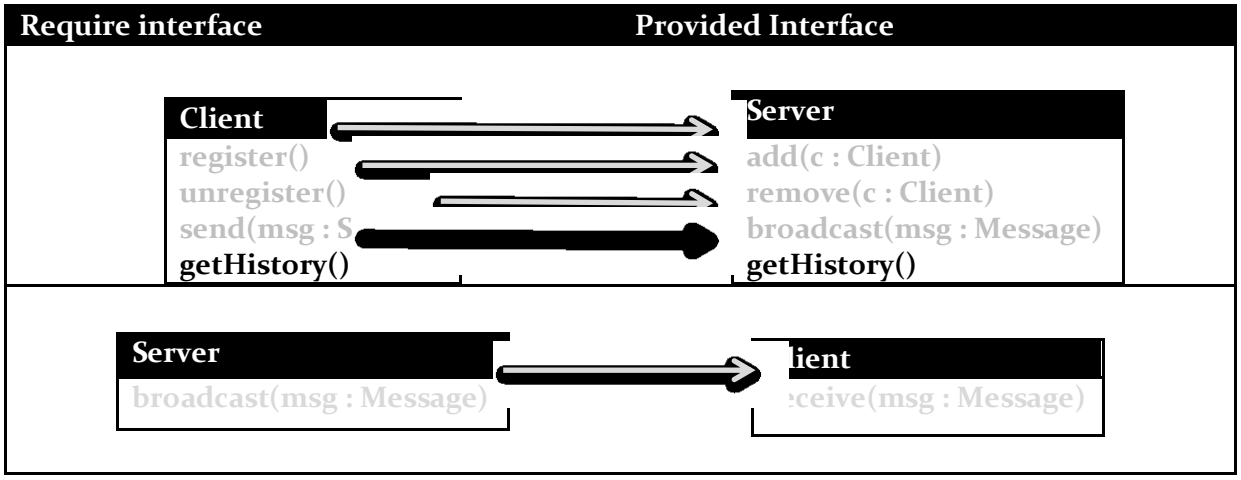
5. History

The chat provides a history feature. At any time a client can load the history of the chat. This history is stored on the server side in an xml file.

To store the history we decide to use an xml file instead of a binary file. The serialization is made with the XMLTool class. It provides two methods, `encodeToFile(Object obj, String fileName)`, `decodeFromFile(String fileName)`.

The first one use to encode an object in to a file, and the second return the encode object from a given file. XMLTools use java.beans.XMLEncoder and java.beans.XMLDecoder to perform this action.

Because the history is loaded only on the client needs, we had to add a new remote call to get the history from the server.



Of course all the necessary changes were done to perform the new remote call.

6. Graphical User Interface

To create the GUI we used the library Swing. We are using the Model View Controller (MVC) design pattern to create the GUI.

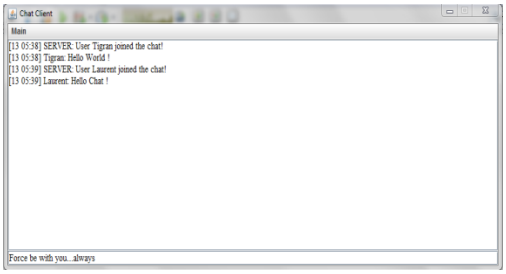


Figure 6-1: The chat

7. Handbook

Step1: Launch the rmi registery

Step2: Launch the server

```
>java -Djava.rmi.server.hostname={ip of the server} -jar Server.jar
{ip of the server}
```

Step3: Launch a console client

```
> java –jar ClientConsole.jar {ip of the server}
```

Step3bis: Launch a gui client

```
> java –jar ClientGui.jar
```