Distributed Systems – Java RMI Chat System
Project documentation

Project documentation content:

Algorithm design

The chat system basically retains the same broad outline of the algorithm on which we had made the previous TCP implementation. However, by using rmi, we are able to reduce workload on the server, which in the TCP version had to analyze each message from clients and parse it into either commands or regular text message.

Interfaces:
The interfaces extends Remote Class.
Server: ServerInterface

The server interfaces provides three following methods.

1. Join the server: join()
2. Send message to other users: send()
3. Disconnect from the server: disconnect()

Client: ChatObjectInterface

The client interface provides a single method, which wraps the old functionality of the previous TCP version.
It basically makes the client display a message or signal it to quit:
handle();

Class
Server: Server

The class Server extends the UnicastRemoteObject and implements the server interface. This is the one and only executable for the server.

Client: ChatObject

As we have realized the main client application as an applet, we had to implement the UnicastRemotObject in this Class. This class wraps around the applet class to provide the remote functionality to the client. The ChatObjectInterface is implemented in this class.

Client: Client applet

Client applet reused the old codes from the TCP implementation. It has been slightly updated to meld with the ChatObject class to provide remote implementation.

Description of sequence / flow chart:

Server

0. Start rmiregistry.
1. Register the server as “RMICHAT”
2. Wait for clients to call its connect() RMI method.
3. This method gives to the server a remote stub of the Client.
4. On successful connection add this client stub to the list of connected clients and send a welcome message to the client using the rmi method of the Client remote object. ChatObject.handle(“Welcome”)
5. Wait for client to call the next server rmi method send(String message) this method

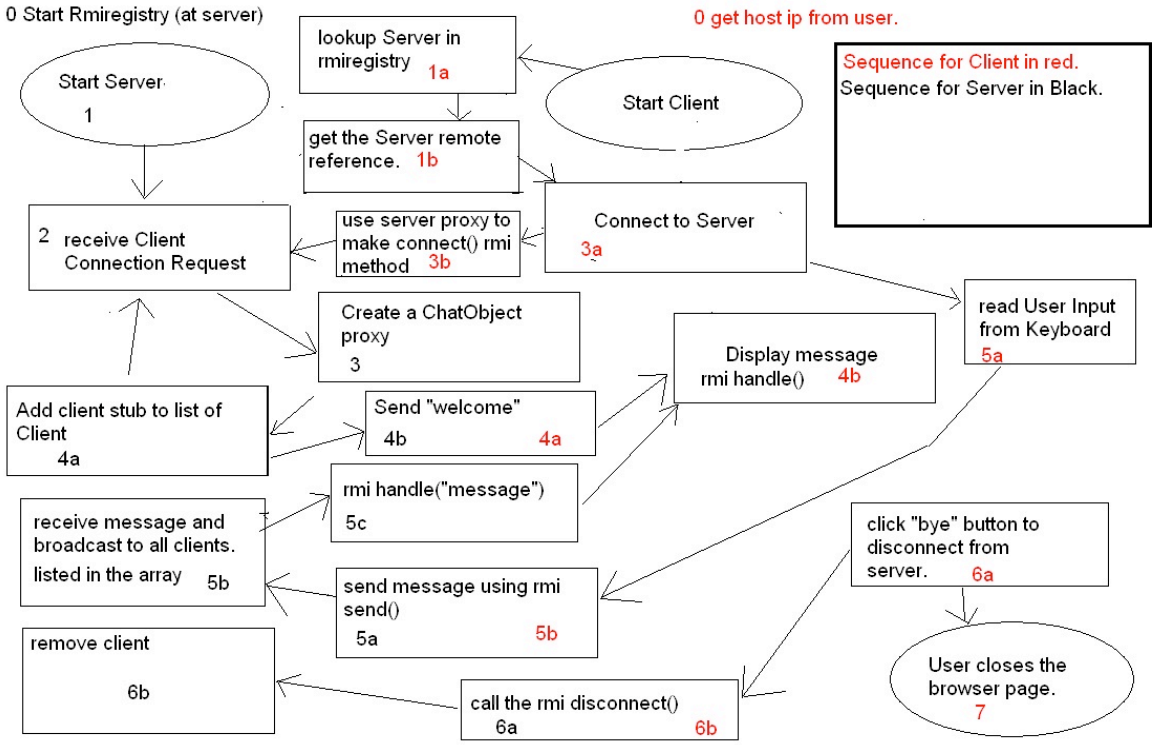
takes the string passed as parameter and broadcast it to all the client registered in the server.

6. Wait for client to call the last server rmi method disconnect() which removes the reference of the remote client in the servers list of client.
7. When the server administrator orders the server to shutdown, close the program. (as courtesy, inform connected client that server will shut down. Not implemented.)

Client

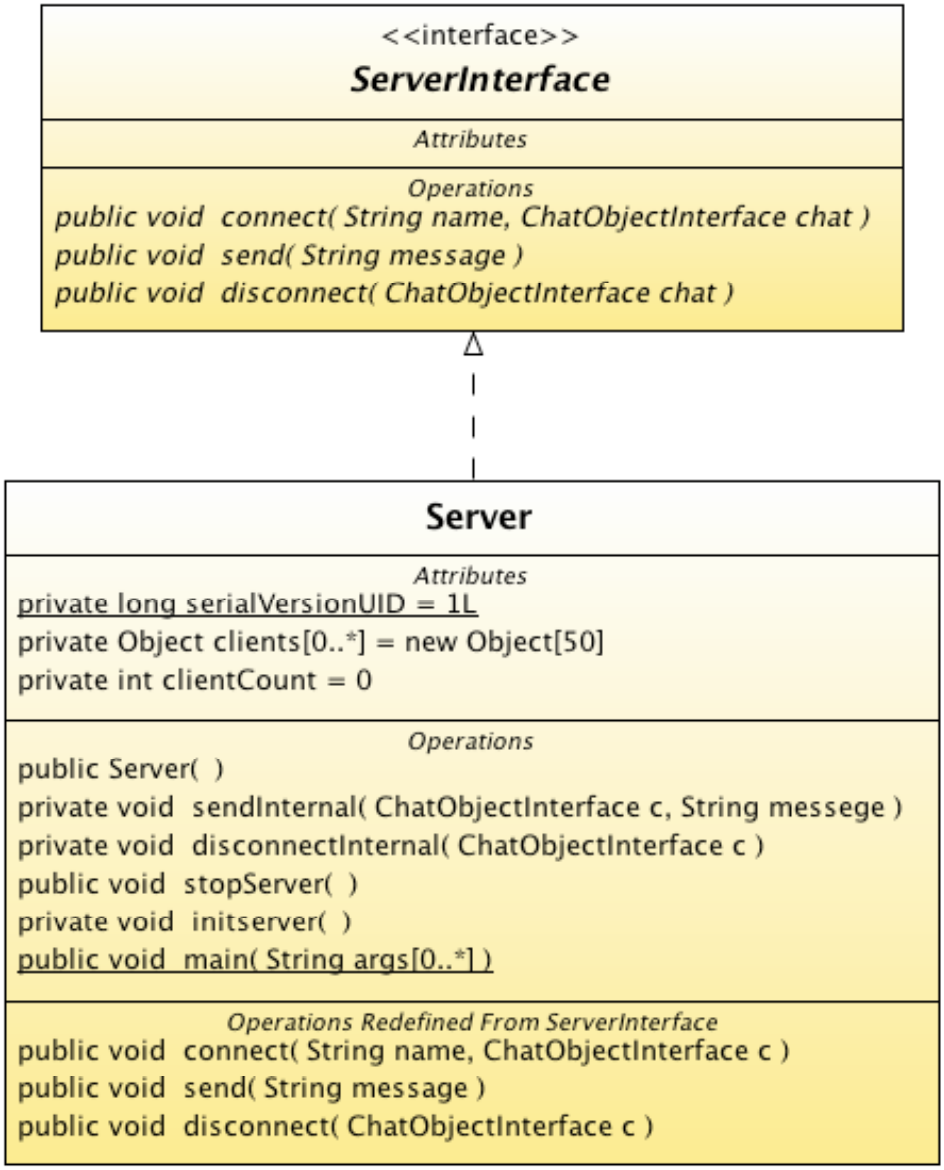
0. Get the host address of the server from the user.
1. Look up and get the server stub using the lookup. This gives the server remote reference to the client.
2. Create a ChatObject. It is the remote object of the client.
3. Call the server rmi method connect() and send its own stub (ChatObject) so that the server can use the methods present locally on the client.
4. The server will send a welcome message to the client if the client has been succesfully connected.
5. The client can now chat with other users. Type the message and click on the send button to call the rmi method Server.send (String message). The effect of this method has been described earlier.
6. The client can disconnect from the server by sending the rmi method Server.disconnect(). On the server side the client stub is removed from the list of clients, which will stop the client from receiving any new message.
7. As our client has been realized as an applet, we simply disable the control of the applet and ask the user to close the web page.

RMI CHAT SYSTEM PROGRAM SEQUENCE GRAPH:

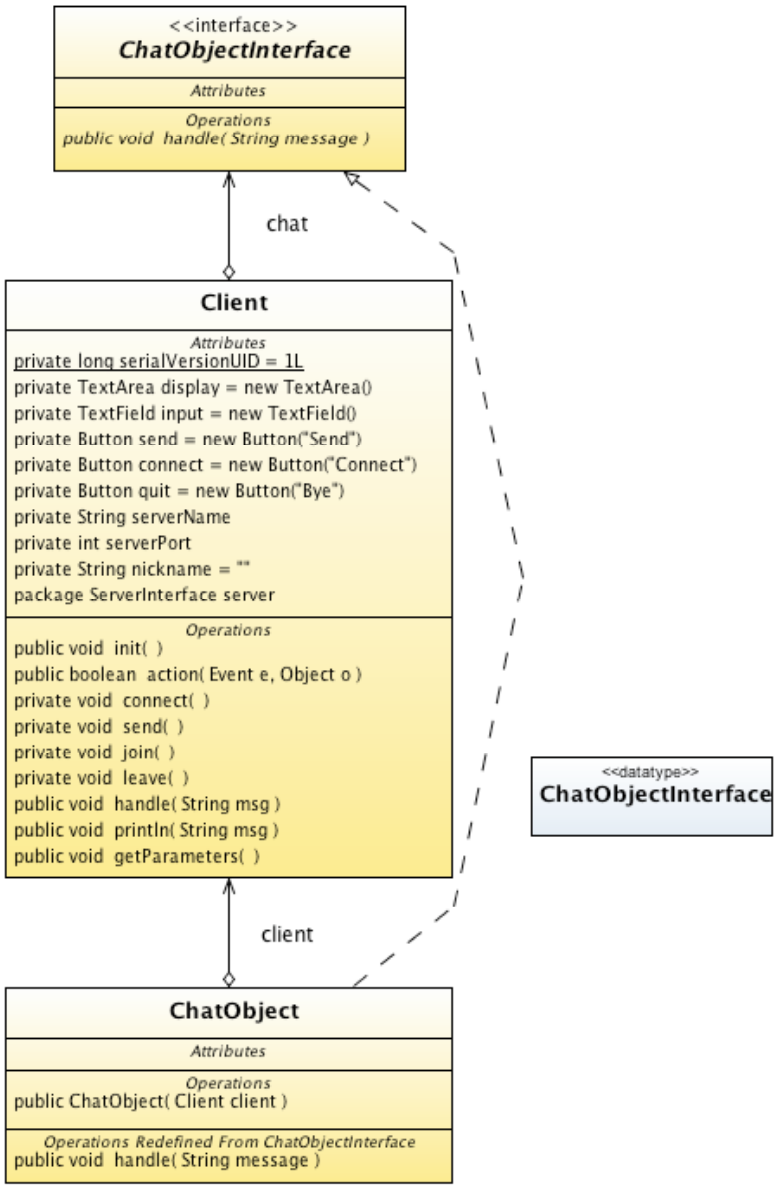


UML diagram:

UML diagram for server:



UML diagram for client:



Classes:

There are the flowing classes being used from the server:

- Server.java
- ServerInterface.java

There are the flowing classes being used from the client:

- ChatObject.java
- ChatObjectInterface.java
- Client.java

Testing:

Deployment

The server successfully runs on Windows XP, kubuntu 8.04, Mac OS X 10.5.5.

The client applet successfully works in Firefox 3 and IE 7 and Safari.

Bugs!

We have no major bugs. The program is functional, and moderately safe. We have used one deprecated methods belonging to the thread class, which results in compiler warnings.

Performance testing idea

We would like to test the server performance in the following way. We define a standard test message, a string of 80 chars. We then modify our client so as to be able to send this test message and confirm that it reaches the server and it is echoed back to the client.

Each client has a unique test message so that it will be able to identify its echo.

The client will send a fixed number of test messages and count how many of the messages is received back, and calculate the time taken by each message.

The test scenario: we start the server.

Then we connect 1,5, 10, 20, 30, 50 clients.

At each population level, we do the test and plot the average response time and reliability in a table, like the following:

No. of Client	Message Loss	Latency

Message loss=(message sent- received)/100

Latency=sum of latency for each message/no of message received

As next we do the same performance test for the TCP implementation of the server.
Finally, we get an idea of the whole performance of both systems.

Differences from the TCP Implementation

The major differences from the programmer's point of view is that the program is much simpler and easier to understand. A beginner java programmer can start implementing this kind of network application without having to deal with low level network i/o like sockets, tcp protocols etc.

Regarding reliability: the RMI abstraction of making the network invisible between the client and the server effects the reliability of the chat service in several ways.

It is debatable whether this overhead in wrapping our service in RMI methods justify the extra coding which may introduce additional bugs. It is also not clear how efficiently and safely RMI can handle multiple threads. Therefor we do not know how well this server will scale up to support larger number of clients reliably. At present we have limited our server to a fixed maximum limit of 50 clients.

Performance:

As can be expected, the RMI abstraction comes at a cost of higher system requirement (call marshalling and so on) then a comparable TCP server. At present , our rmi methods directly interact with the server object and client object. This is significantly slower then sending simple text message as was done in the TCP server. We are aware that a better and more efficient way of implementing the server would be to think of the server as a network printer. It would spool or queue the message, and at the same time , a second thread will take message from the queue and send it to all the client. On the client side, there could be a local message-in queue, which will receive message from the servers second thread. This enable the server to return quickly from the rmi method of sending message to client. On the client side, the rmi method to send message o the server will also return quickly (as it has only to enter the message in the queue) and a similar second thread would display the message in the queue. These proposals are aimed at reducing the time on the rmi methods, as they take more time to execute then local methods. This way our performance gap between the rmiServer and TCPserver would be minimum.

Installation and Usage Guide:

For running the server please follow these instructions:

Usage of the Server:

- Use the command line and be sure that you are in the same folder where you find also this documentation, the client.html and the folder for the client and the server.
- Then you can launch the RMI registry with the following command (after that nothing special happens but please don't close this terminal window while running the server):
  - `rmiregistry`
- Open a new terminal window and type the following command for starting the server:
  - `java rmiServer.Server`
- If you get a pop-up window in which is written "Server running, press yes to stop" the server started correctly.

- You can stop the server click on the "Yes" button, then the program terminates.

- As next you can also terminate the rmiregistry by changing to the terminal window where the rmiregistry was started and type ^C.

Usage of "client.html":

- Use your web browser to start the client. Be sure that the "client.html" file is in a folder that has a subfolder "rmiClient" that contains the necessary files for launching the client.
- Then you can launch the client by clicking on "client.html" or explicitly launching it by your web browser (if not automatically opened).
- As next you are asked to enter the IP of the server and your user name.
- Then the applet gets launched and you can connect finally to the server which you have entered by clicking on the "Connect" button. Be sure that the server is running if your are testing it on your local machine.
- Now you are able to send messages by typing it into the empty field and clicking on "Send" button for sending it.
- For leaving the chat you can click on the "Bye" button and then close the browser window.