Assignment 2 Design Document

GUI View Design

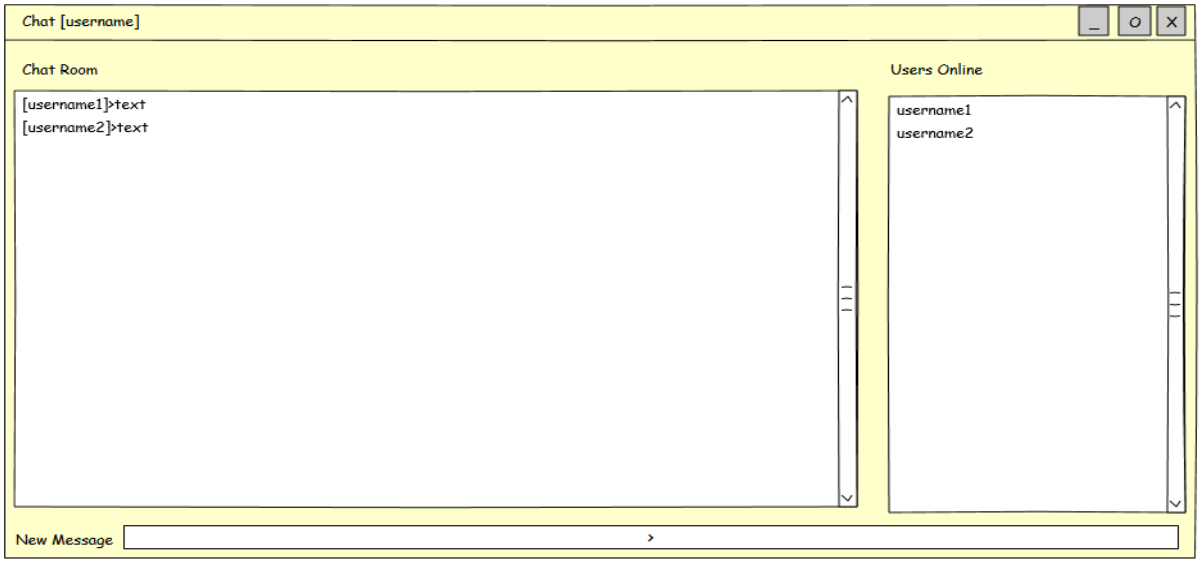

Figure 1 GUI Design

Chat GUI View should have 3 main parts:

1. *Char Room text pane:* where user will see all the messages from all connected users appearing from bottom up. Latest message will appear on top of the pane.
2. *Users Online text pane:* where user will see all the connected users
3. *New message text field:* where user can write a new message and on ENTER message will be send to server and populate on every connected user Chat Room text pane

After some research done it seems that chat rooms messages have their standard format.

```
[icon][username]>message text
```

Where icon and username is chosen by user at client application startup. Username is given random color at startup as well. For this to function I will need to pass these lines to the server.

For this I should create an immutable java bean which will represent this line format and be passed between the clients and the server. For an object to be pass to a RMI stub method it must implement the Seriazable interface, because RMI uses object serialization over TCP socket.

```
public interface Line extends Serializable{
    Icon getIcon();
    Color getColor();
    String getName();
    String getText();
}
```

Figure 2 Line Interface

GUI view interface

```
public interface ChatView {
    void print(Line line);
    void print(List<Client> onlineClients);
}
```

Figure 3 GUI interface

Where *print(line)* method will insert given line in the Chat Room pane and *print(onlineClients)* method will fill the Online Users pane with clients names and icons from *onlineClients* list. This way I should be able to implement different kinds of Views as long as they comply with the interface. I will build one GUI view using *javax.swing* library.

Chat Server interface

For a client to be able call a server's methods using remote method invocation a server must implement an interface which inherits from *Remote* interface with every method throwing *RemoteException*. The server interface implementation must also inherit from *UnicastRemoteObject* class.

```
interface Server extends Remote {
    void connect(Client client) throws RemoteException;
    void disconnect(Client client) throws RemoteException;
    void send(Line line) throws RemoteException;
}
```

Figure 4 Chat Server interface

Where *connect(client)* method will register the client. Method *disconnect(client)* will unregister the client, and *send(line)* will send a chat line to server to process.

Chat Client interface

For the server to call a client's methods remotely a client must implement an interface which inherits from *Remote* interface, with every method throwing *RemoteException*. The client implementation must also inherit from *UnicastRemoteObject* and to be seriazable which Remote already is. That is because the client object will be passed to server stub methods as a method parameter.

Client will be just a proxy for GUI interface plus will provide the getters for icon, color and username for server to have access to them if needed.

```
public interface Client extends Remote {
    void print(Line line) throws RemoteException;
    void print(List<Client> connectedClients) throws RemoteException;
    Icon getIcon() throws RemoteException;
    Color getColor() throws RemoteException;
    String getUserName() throws RemoteException;
}
```

Figure 5 Chat Client interface

How this all will work?

It will be a simple observer pattern implementation using java RMI technology.

A client will remotely invoke *server.connect(client)* stub method which will add the client into a connected clients collection on the server. Then on *server.send(line)* method invocation the server will pass this line to all clients from the collection. On *server.disconnect(client)* stub method call the server will remove the given client from the collection. Access to all three server stub methods implementations should be *synchronized* because they can be invoked by many clients at the same time and they modify the server state.

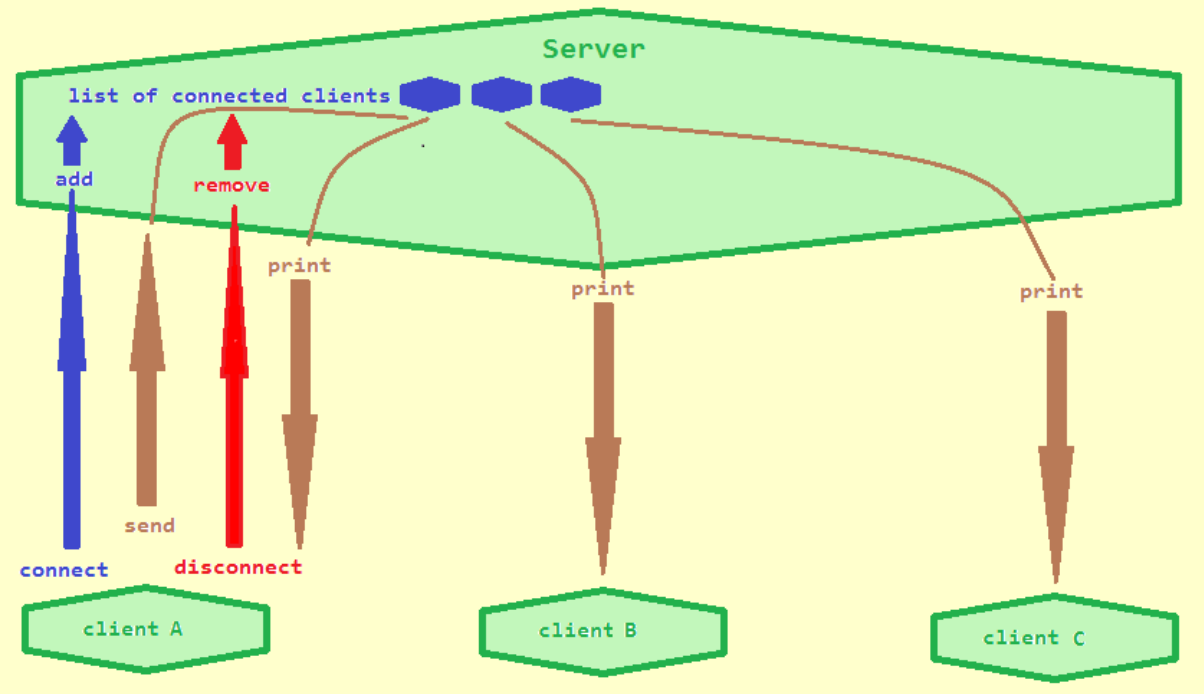

Figure 6 Observer pattern visualization

Conclusion

The proposed design was proven quite solid. Application was developed smoothly without any serious problems and without a significant deviation from the design. Application works well without the bugs on the localhost.

The challenge lies in making this work over an actual network.

I managed to run it on Windows Domain on LAN.

I have tried to run this between three *Linux* boxes in *Virtual Box*, but for the some kind of network misconfiguration or misunderstanding of the involved technologies I'm getting various RMI related exceptions. More research and testing must be done here.

I tried to run chat server on virtual machine in Azure Cloud, but the connection always timed out.

Additional functionality implemented:

- The chat server is remembering last 20 lines of the chat lines, so when a new client is connected he sees last 20 lines of ongoing conversation.

- Security manager and custom policy file implemented.
- Chat view incoming message sound implemented.

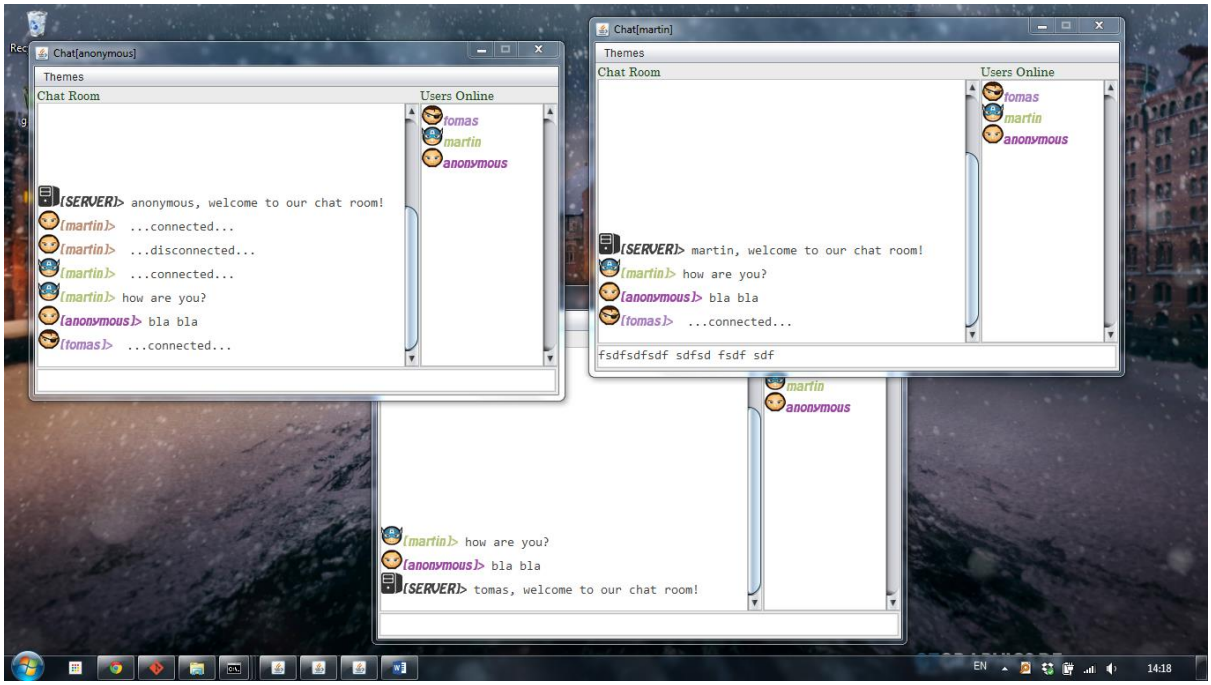

Figure 7 Chat Application implementation