Client-Server Based Chat Application Using Socket Programming

George-Catalin Muselin, Simeon Tudzharov and Taiwo Razaq
COMP1549: Advanced Programming
University of Greenwich
Old Royal Naval College
United Kingdom

***Abstract*—This paper goes through the creation of a Client-Server based application using socket programming and discusses in detail the steps for making it work and being deployed. Network systems are built to communicate with each other and made possible through service-oriented architectures. The Client-Server architecture is used to develop the chat application. A Chat application is realized for the Client and the Server which is biased on Transmission Control Protocol (TCP) where TCP is the connection oriented protocol and is reliable. After starting a server on a specific port, users can establish a connection to it through a friendly Graphical User Interface mentioning the port and the IP address of the creator. The first user connected to the chat becomes the coordinator of that chat and is put in charge of maintaining the conversation and pinging the state of the group to all other members. On log out the title is lost and is inherited by other user.**

Key words: *(socket programming; Client-Server; Java NetBeans; TCP; IP address)*

I. Introduction

Nowadays, Client-Server programs are no longer an unfamiliar concept in distributed computing and in computer networks. In fact, people use chatting programs on a daily basis, to either communicate or just to keep up with the trends.

Typically, several network systems are built to communicate with each other and are created to be available through service-oriented architectures. In this project, the main focus was the Client-Server architecture used to create and develop a secure and trusted chat application in which people can communicate easily and freely. The chat application is designed based on Transmission Control Protocol (TCP), which represents the connection oriented protocol and, multithreading to establish the application.

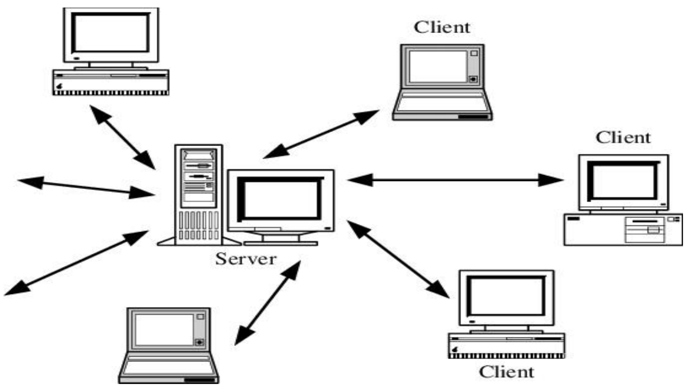


Figure 2.2: Illustration of a Client-server network

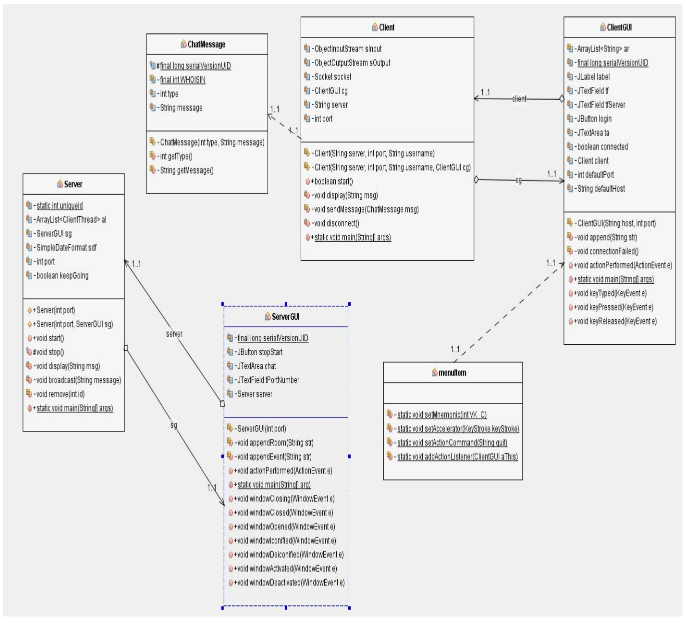
The Client-Server chat application consists of a Client and a Server and there exists a two way communication between them. A Client is a program that allows users to access and view its interfaces. Every single Client communicates and connects with each other through a server. In an overall concept, the Client delivers as an initiator in communication and the server is responsible of responding to the Client`s request after waiting passively. Normally, the server process will start on a computer system and it must be executed before the Client. After the server has finished providing its service to the Client, it will go back to sleep, waiting for the next Client request to reach. This process is repeated as long the server processes are running.

II. Design/implementation

The program was designed and created using socket programming in Java NetBeans supported by TCP. The Client server based program has different functions and utilities that were analyzed, demonstrated and simulated. The different architectural elements within the system are presented in the following sections.

The Protocol

The protocol established between the Client and the server is determined in the ChatMessage class. In its constructors, there are different contracts created for different instances of the ChatMessage class. First of all, each message has a type that notifies the server of the request type and prompts relevant response. These are the fields of the ChatMessage class and are listed as final integers. There are five types of messages that can be sent between the Client and the server: login, chat, logout, error, who is in.



ChatMessage
#final long serialVersionUID
~final int WHOISIN
-int type
-String message
~ChatMessage(int type, String message)
~int getType()
~String getMessage()
Client
-ObjectInputStream sInput
-ObjectOutputStream sOutput
-Socket socket
-ClientGUI cg
-String server
-int port
~Client(String server, int port, String username)
~Client(String server, int port, String username, ClientGUI cg)
+boolean start()
-void display(String msg)
~void sendMessage(ChatMessage msg)
-void disconnect()
+static void main(String[] args)
ClientGUI
~ArrayList<String> ar
-final long serialVersionUID
-JLabel label
-JTextField tf
-JTextField tfServer
-JButton login
-JTextArea ta
-boolean connected
-Client client
-int defaultPort
-String defaultHost
~ClientGUI(String host, int port)
~void append(String str)
~void connectionFailed()
+void actionPerformed(ActionEvent e)
+static void main(String[] args)
+void keyTyped(KeyEvent e)
+void keyPressed(KeyEvent e)
+void keyReleased(KeyEvent e)
Server
-static int uniqueId
-ArrayList<ClientThread> al
-ServerGUI sg
-SimpleDateFormat sdf
-int port
-boolean keepGoing
+Server(int port)
+Server(int port, ServerGUI sg)
+void start()
#void stop()
-void display(String msg)
-void broadcast(String message)
~void remove(int id)
+static void main(String[] args)
ServerGUI
-final long serialVersionUID
-JButton stopStart
-JTextArea chat
-JTextField tPortNumber
-Server server
~ServerGUI(int port)
~void appendRoom(String str)
~void appendEvent(String str)
+void actionPerformed(ActionEvent e)
+static void main(String[] arg)
+void windowClosing(WindowEvent e)
+void windowClosed(WindowEvent e)
+void windowOpened(WindowEvent e)
+void windowIconified(WindowEvent e)
+void windowDeiconified(WindowEvent e)
+void windowActivated(WindowEvent e)
+void windowDeactivated(WindowEvent e)
menuItem
~static void setMnemonic(int VK_C)
~static void setAccelerator(KeyStroke keyStroke)
~static void setActionCommand(String quit)
~static void addActionListener(ClientGUI aThis)
client
cg
server
sg
1..1

Figure 2.3: Illustration of the program`s UML

The Client

The Client is responsible for communicating user requests to the server and informing the user of the response sent by the server. This suggests a two-way communication: the Client wraps up messages that are sent to the server and unwraps the messages sent by the sever. Each message created by the Client receives a type as explained in the protocol section; moreover, each will have a time stamp and a text field with information depending on the type of message that is being sent. To interpret messages received from the server, the Client first checks the type of the message and takes appropriate action in response.

The structure of the Client is given by the two main classes: Client and ClientGUI.

The Client class is the backend that contains all the logic and handles the communication with the server.

Sending

Depending on the user request, the client may send four types of ChatMessage objects to the server.

1. **A login request,** including just an name(string) or an ID(integer). The first user logging in to the Server becomes the *coordinator* of that Server. The program does not support encryption making it easily to set up a server and chat with other users.

2. **A chat request,** in which the ChatMessage object is wrapped with the time stamp, time, the sender of the message and the text message itself. After all, the ChatMessage object is then sent to the Server.

3. **A logout request,** this simply wraps up the type, a non-meaningful String to inform the server and the time stamp that the user wishes to be logged out. When logging out, if the user that logged out was the *coordinator* of that server, the title gets passed to the second user that logged in on the Server making him the new *coordinator*.

4. **A Who is in request,** which cycles through the users connected that are stored in an array and prints out in the main text field the type, time stamp, the username and the title (if the user has one) of all the connected users.

Receiving

Depending on the response from the server, the client may receive three types of messages from the server. The type of the different message is determined as set out in the protocol.

1. **A login response,** which updates the model to reflect successful login.

2. **A chat response,** where the client is going to display the incoming message.

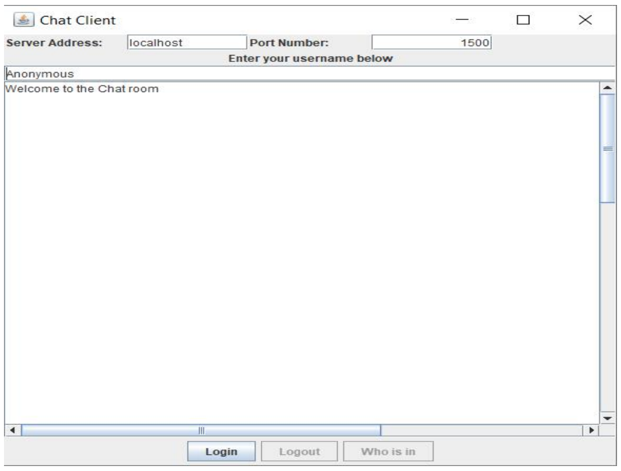
3. **An error message,** which is displayed to the user.

The ClientGUI is the front end of the program and it ensures user-friendly communication with the rest of the components. It uses a number of models and views to inform the user of the client/server activities:

1. **Login view:** includes the text filed in which the user is gonna type his name or ID and a Login button that establishes the connection to the server and enables all the other buttons. As a default the username will be “Anonymous” Following up, users get a welcome message in the chat.
2. **Chat view:** includes the chat window that is updated by the client model and chat model.

3. **Chat model:** updates the chat view with new messages.

4. **Client model:** updates the chat view with new messages.



Chat Client

Server Address: localhost Port Number: 1500

Enter your username below

Anonymous

Welcome to the Chat room

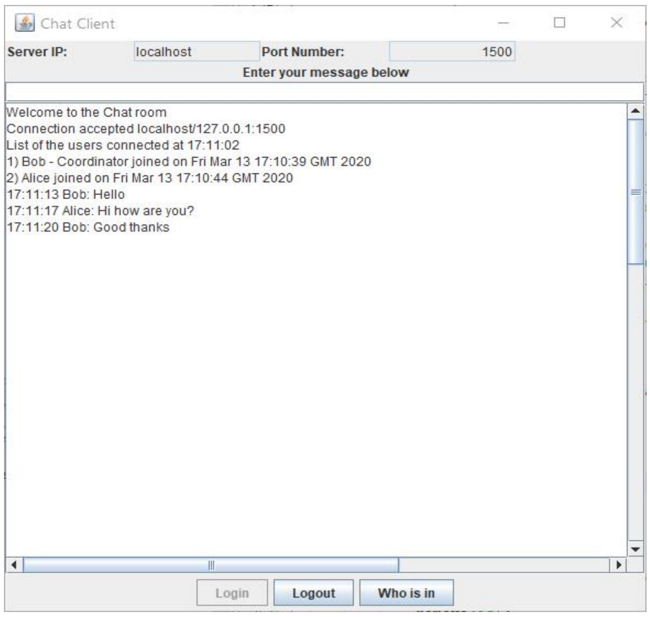
Login Logout Who is in

Figure 2.4: Illustration of the ClientGUI on run

5. **Who is in view:** includes the list of online users along with the time stamp and the title.

6. **Port Number selection:** the default port is set to 1500, but users can change it accordingly

7. **Server Address selection:** the default address is set to localhost, but users can change it accordingly



Chat Client

Server IP: localhost Port Number: 1500

Enter your message below

```
Welcome to the Chat room
Connection accepted localhost/127.0.0.1:1500
List of the users connected at 17:11:02
1) Bob - Coordinator joined on Fri Mar 13 17:10:39 GMT 2020
2) Alice joined on Fri Mar 13 17:10:44 GMT 2020
17:11:13 Bob: Hello
17:11:17 Alice: Hi how are you?
17:11:20 Bob: Good thanks
```

Login Logout Who is in

Figure 2.5: Illustration of two users chatting

The server

The server is responsible for receiving requests from the client and passing it to another client. Similarly to the client, it unwraps the ChatMessage object received from the client and checks its type. After that, it goes on to fulfil the request and wrap up a ChatMessage object as a response to the client

<u>Sending - Receiving</u>

Depending on the user request delivered by the client the server may receive several types of ChatMessage objects from the client.

1. A log in request, which accepts the user in the system

2. A chat message requested, which is delivered to the recipient.

3. A logout request, which logs the user on the given client off the system.

The Server class is the back end of the program and contains all the logic making it possible to start the server using TCP and socket programming.

Socket programming is a two way communication link between a Client and a Server that are running in a network environment. It has an efficient communication algorithm between two or more computers.

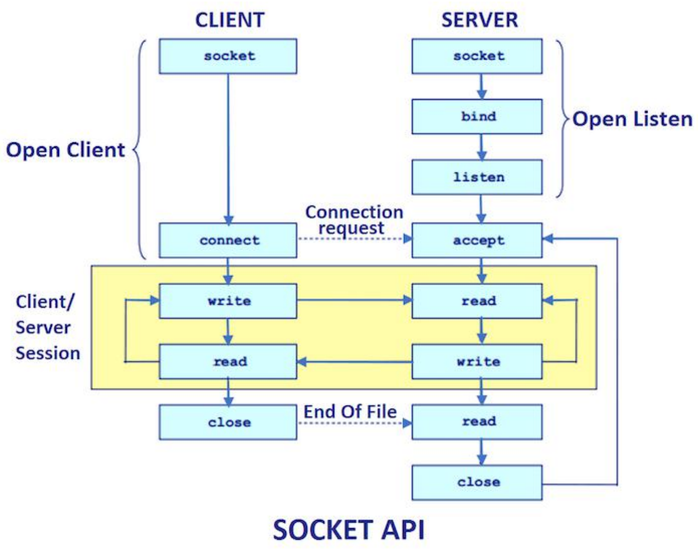
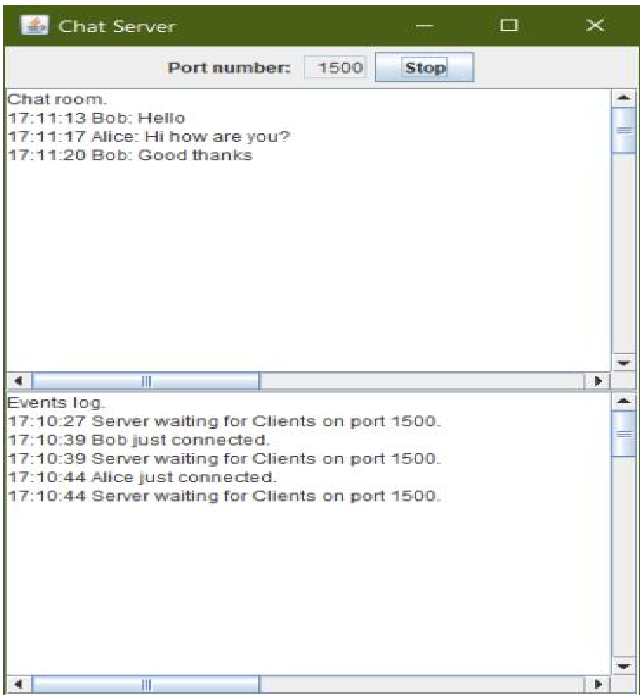


Figure 2.6: Illustration of Socket Programming

Java provides two classes for this purpose: Socket and ServerSocket. The sockets connect two programs and implement two sides of the program (the Client and the Server).

The ServerGUI is the front end of the program and it ensures user-friendly communication with the rest of the components. It uses a number of models and views to inform the user of the client/server activities as explained in the ClientGUI.



```
Chat room.
17:11:13 Bob: Hello
17:11:17 Alice: Hi how are you?
17:11:20 Bob: Good thanks
```

```
Events log.
17:10:27 Server waiting for Clients on port 1500.
17:10:39 Bob just connected.
17:10:39 Server waiting for Clients on port 1500.
17:10:44 Alice just connected.
17:10:44 Server waiting for Clients on port 1500.
```

Figure 2.7: Illustration of the ServerGUI class interface

Analysis and Critical Discussion

To ensure the Chat Program functions and ensures the user a good environment to chat, a test plan was conducted. The objective of this test plan is to lay out the testing strategies that will be used to test the different components of the Chat Server system. Where applicable, it will also include the predicted results from each test.

The test covered all the classes and any additional components and those were tested in isolation to ensure not causing any bugs or errors.

Once each component has been thoroughly tested in isolation then an integration testing was performed to ensure that the components interact with each other as expected. The testing covers the link between Server-Client and ServerGUI-ClientGUI. This concluded with a full system test as some processes such as logging in, required communications to be passed along multiple links and back. In the end, the system performance was tested to make sure that it meets the certain performance requirements. The most important part of this test was CPU usage and time lags, as it represents a big concern and it is important for usability that both of these are kept as low as possible.

Unit Testing

To ensure that the system worked as expected, each method in all the classes has been tested accordingly using JUnit.

1. Functional Testing

Server-Chat application module
The server chat application module consists of two components which enable the module to work effectively, namely the port, IP address.

A. **Server chat Port:** The port window has been tested using the client module`s port number and connection was established successfully. An invalid port number inputted into the window notifies the user that the port is wrong and a working port must be inputted in order to connect.

B. **IP address:** The IP address window has been tested using the client`s module IP address and the connection was established successfully. Providing an invalid IP address will result in an error message outputted to the user.

2. Stress Testing

This type of test enables the developed software to determine the stability of the system. Whenever the coordinator logs out, the title will be passed to another user and the communication will still be possible until the Server is closed down. If the server is closed, all the users are notified in the main text field.

3. Structural Testing

This test examines the internal processing logic of the developed program. However, the prototype of the server chat module is able to perform its intended tasks for example messaging. After a server is started, on a specific port, users can connect on that port by running ClientGUI and changing the default (1500), same works for the IP address (default is localhost). After the IP address and the Port number have been imputed and the user has written an ID, on clicking log in they have connected to the chat. Every member of the chat knows of everyone else online at that moment by clicking the “Who is in” button. This will display all the users and coordinator along with the time stamps. The coordinator also pings every now and then the status of the chat group, including IP addresses, time stamps and user list. When an user wants too disconnect they can click on the "Log out" button and will be out of the server. All the users can close the program by simply pressing "Ctrl+C".

Client-Chat application module

Apart from the tests done in the Server-Chat application, the following test has been done:

1. Integration Testing

This is a type of software testing in which the modules of the developed program are combined and tested as a group. This test has the ability to ensure that both of the program modules are communicating with each other effectively.

A. Top-down testing

Both the Client and the Server modules have been tested based upon connection establishment where the connection is initialized from the server to the client using their port numbers and IP addresses. Both the server and the client modules are able to exchange instant messages.

B. Bottom-up testing

The client and the server modules have been tested where connection is established from the client to the server using their port numbers and IP address. Instant chat messaging returned with success.

III. Conclusions

The Chat program in its final state successfully provides a stable and trustworthy application in which people can chat with each other without any encryption or database. It is secure and handled very carefully so it can provide the best User Interface and User Experience for the people willing to chat. Also, the chat application, is really easy to use and does not require any networking or vast programming knowledge. Resource wise, the program is small and dynamic and yet it manages to provide everything an user would want. The program is developed so that the first user that connects to the server becomes the coordinator and is in charge of maintaining the chat. Continuing to that, the coordinator pings the state of the chat occasionally to ensure every member is aware of the others and online. However, there is a button made for everyone to check who is online at the same time with time stamps and title.

Overall the Chat program it is doing its most important task: to keep people connected together in a friendly environment and to ensure a great communication.

References

1. A distributed, value-oriented XML Store - Scientific Figure on ResearchGate. Available from: https://www.researchgate.net/figure/Illustration-of-a-Client-server-network_fig2_2605250 [accessed 12 Mar, 2020]
2. Java Socket Programming. Available from: javatpoint.com/socket-programming [accessed 12 Mar, 2020]
3. Unit Testing with JUnit - Tutorial. Available from: vogella.com/tutorials/JUnit/article.html [accessed 12 Mar, 2020]
4. Creating a Chat Server Using Java. Available from: https://www.instructables.com/id/Creating-a-Chat-Server-Using-Java/ [accessed 12 Mar, 2020]
5. Design Patterns for Beginners With Java Examples. Available from: https://dzone.com/articles/design-patterns-for-beginners-with-java-examples [accessed 12 Mar, 2020]
6. Socket Programming in Java. Available from: https://www.geeksforgeeks.org/socket-programming-in-java/ [accessed 12 Mar, 2020]
7. Java - Networking. Available from: https://www.tutorialspoint.com/java/java_networking.htm [accessed 12 Mar, 2020]
8. Java Socket Client Examples (TCP/IP). Available from: https://www.codejava.net/java-se/networking/java-socket-client-examples-tcp-ip [accessed 12 Mar, 2020]