ChatBridge

Author 1 name and Author 2 name
COMP1549: Advanced Programming
University of Greenwich
Old Royal Naval College
United Kingdom

***Abstract*— The client-server chat application using Java is the main topic of this research paper. Chat bridge is the name of the said application. The architecture of the proposed chat application and the network protocols employed to establish communication between the client and server components are among the technical aspects of developing a scalable and reliable chat programme that are covered in the paper. The specifics of the implementation, including server calls, user interface design, client calls, junit testing, and the use of Java libraries for network programming, are covered. The design also implies singelton design pattern in order to make sure that only one instance of server can be running at a specific time. The paper's discussion of the value of developing best practises and standards for dependable chat systems that meet modern communication needs serves as its conclusion. This applies to both the client-side and server-side GUI implementations. A connected client can send both broadcast message and private message to other clients.**

Keyword: *Client Server, Socket Programming, Chat, muti threaded application*

I. Introduction

Client-server chat application is a network-based software system that enables users to exchange real-time messages over the internet. The system's goal is to make it easier for two or more clients to communicate with one another. These clients are connected to a main server, which acts as a kind of intermediary. These programmes are frequently used for both personal and professional communication, and usage has greatly increased recently.

The creation of a client-server chat application in Java, one of the most popular programming languages for network-based applications. Designing and implementing a scalable, dependable chat programme that can support numerous clients at once is the aim of this research.

The technical aspects of the client-server chat application and the many network protocols that can be utilised to construct such systems will be covered in the opening section of the article. The architecture of the suggested chat application, including the layout of the client and server components and the network protocol used to establish communication between them, will next be described in depth.

The research will also cover the implementation details of the chat application, including the use of Java libraries for network programming, user interface design, and testing methodologies. Ultimately, this research paper will give readers a thorough grasp of client-server chat application design and execution, stressing the important factors to take into account and the difficulties that arise while creating such systems. It will also aid in the creation of best practices and standards for creating dependable chat systems that are scalable and satisfy the demands of contemporary communication needs.

II. Implementation

This project is a simple chat application constructed using Java programming language. The application runs on a Server-Client architecture, where multiple clients can connect to the server to communicate with each other.

Server class

The server-side implementation is designed using Java's socket programming API(Application programming interface), which provides a set of classes that allow communication between two machines. The server creates a ServerSocket and listens for incoming connections from the clients. Once a client connects, the server accepts the connection and creates a new thread to handle communication with the client.

The server implementation uses a ServerUtility class, which represents a connected client. This class holds the client's socket connection and a unique identifier assigned by the server. The server also maintains a list of connected clients in an ArrayList. Whenever a new client connects, the server creates a new ServerUtility object and adds it to the list of connected clients. So all the clients currently connected with the server must have entry in the said ArrayList.

The application allows clients to send private messages to specific clients, as well as broadcast messages to all connected clients. To achieve this functionality, the server implementation includes methods for sending private messages and broadcasting messages to all clients.

To handle private messages, the client sends a message containing the receiver's ID and the message content. The message is parsed by the server and forwarded to the designated client. Broadcasting messages is achieved by iterating through the list of connected clients and sending the message to each client.

To ensure that messages are received by all clients, the server maintains a coordinator that is responsible for managing the communication among the clients. The coordinator is selected from the list of connected clients, and any client that is not responding is removed from the list. The server updates the coordinator whenever the list of connected clients changes.

Server utility class

The provided code is a class called ServerUtility that extends the Thread class in Java. It is used in a server-client communication model to handle communication between the server and the client. The class includes a constructor that initializes the socket, input and output streams, and client ID.

The run() method of the class creates a new thread to receive messages from the client. When a message is received, it calls the received() method of the class passing the message as a parameter. The method then checks if the message is either a client left handler or a broadcast message, and if so, calls the appropriate method from the Server class.

The sendMessageTo() method takes two parameters: the client ID and the message. It then iterates through all connected clients in the Server class and checks if the client ID is either equal to the ID of the current client being iterated or -1 (broadcast message). If it matches the ID, it writes the message to the output stream of that client.

The received() method takes a message as a parameter and first checks if it is a client left handler or a broadcast message. If it is a client left handler, it calls the removeNotRespondingClients() method of the Server class passing the IP address of the disconnected client. If it is a broadcast message, it calls the sendBroadCastMessage() method of the Server class. Otherwise, it assumes it is an individual message and calls the sendMessageTo() method of the Server class passing the message as a parameter.

Overall, the ServerUtility class is responsible for handling communication between the server and the clients and relaying messages to the appropriate clients. It does so by utilizing input and output streams, client IDs, and method calls to the Server class.

Moreover each client has a arraylist which stores all the messages either broadcast or private. So each client have a record of received messages.

ClientGUI class

This section is the implementation of a graphical user interface (GUI) for a client-side application, that connects to a server over a network and allows users to exchange messages. The ClientGUI class is the main class for this application. It extends the javax.swing.JFrame class and implements the WindowListener interface, which allows it to respond to events related to opening, closing, and other actions on the GUI window.

The constructor of the ClientGUI class is used to initialize the GUI components and also sets the default IP and port. Then the connect method is called when the "Connect" button is pressed. It creates a new Socket object with the given IP address and port number, initializes the input and output streams, and sends the client's suggested ID to the server. If the connection is successful, the "Connect" button and IP and port text fields are disabled, and the message type, send message, and send buttons are enabled. A separate thread is created to receive messages from the server.

The actOnRecivedMessage method is called whenever a new message is received from the server. It checks the type of message and updates the GUI accordingly. If the message is a client list update, it extracts the list of available clients and updates the drop-down menu for selecting the message recipient. If the message is a confirmation of the client's ID, it updates the myID variable and the label displaying the client's ID. Otherwise, it appends the message to the chat text area.

The btnSendActionPerformed method is called when the "Send" button is clicked. It constructs the message to be sent based on the selected recipient and message type, and sends it over the output stream. If the message text field is empty, it displays an error message.

The btnConnectActionPerformed method is called when the "Connect" button is clicked. It validates the IP address, port number, and client ID entered by the user, and calls the connect method to establish a connection to the server. If any of the fields are empty or the port number is invalid, it displays an error message.

III. Analysis

The server uses multi threading to handle multiple clients. When a new client connects to the server, the server creates a new thread for the client and adds the client to a list of connected clients. When a client is removed, the server removes the client from the list of connected clients and broadcasts an update to all other clients.
When a client is connected with the server and tries to send message a formatted string is generate containing the client remote address, receiver details and the message. Server then receives this string and checks it is a message to display or a client quit request.
The message is then forwarded to the relative function upon seeing the receiver side. For a broadcast message the format will be

“/127.0.0.1:47526---B---I am online”

This string is then splitted on the bases of “---” which will always leaves three token strings , first index will the address of the sender i.e “/127.0.0.1:47526”second index will points to the receiver i.e if the receiver is B it means it is broadcast message and if it contain id of a client then it will be a private message. And the third index will the actual contents of the message in the above example actual message will be “I am online.
Same is the case with the private message and the string to be transmitted in case of private message will be

/127.0.0.1:47526---client 1---I am online

When a client leaves it leaves the following type of string
--/127.0.0.1:47526
Client can terminate it self by closing its window or by pressing the close button on the GUI besides the connect button. Both the function will call the close window function on the client side which using window listener sends details to the sever side.

IV. Conclusions

Sending private and broadcast message using charbridge is a flawless experience. Moreover the unit tests written also confirm that the application is in perfect working condition. Application uses Java SE library which is widely used for desktop application. As of for the future work one thing can be done is to add a feature of message encryption using some highly secure algorithm like DES or AES. when then can be decrypted on the receiver end.

References

[1] Kalita, L. (2014). Socket programming. *International Journal of Computer Science and Information Technologies*, *5*(3), 4802-4807.
[2] Sawant, A. A., & Meshram, B. (2013). Network programming in Java using Socket. *Google Scholar*.
[3] Mahmoud, Q. H. (1996). Sockets programming in Java: A tutorial. *JavaWorld Online Journal*.
[4] Szeder, G. (2009, July). Unit testing for multi-threaded java programs. In Proceedings of the 7th Workshop on Parallel and Distributed Systems: Testing, Analysis, and Debugging (pp. 1-8).

[5] Lewis, B., & Berg, D. J. (2000). *Multithreaded programming with Java technology*. Prentice Hall Professional.