

Written Report

The networked application created using sockets is a chat application, more specifically an IRC chat using the Java language, created on the IntelliJ IDE. Multiple users can connect to the IRC chat room and enter messages; users who enter are required to assign themselves nicknames before entering any messages into the chatroom; this is done to distinguish users from each other. When a new user connects to the server it will notify all users who had connected beforehand of their entrance.

Commented [AG1]: Not as far as I can see.

Commented [AG2]: This does work

How it works

There are three classes which exist – Client, ClientHandler and Server. The server class is sort of like the host of the IRC room, the code determines how the IRC room itself – not the clients will behave when certain actions occur such as clients connecting.

The ‘Client’ class represents an individual client, multiple clients can be active at once – one of the reasons why IntelliJ was chosen as the IDE was to have multiple clients active at once during testing – clients are told to enter a nickname for themselves, the nickname is based on said client’s user’s own input of characters in response to being told by the ClientHandler to enter a nickname; subsequent messages by the client will be logged to the server, and will appear to other clients as having come from the user associated with the nickname specified.

The ClientHandler class does as its name suggests – it handles the client, in a way bridging the gap between Server and Client, saying how the client should be treated by the server itself. What messages to send when a client joins and leaves for example.

Practices used

Synchronous processing is used here because of the nature of the program as an IRC chat – clients conversating in a traditional back-and-forth manner. Synchronous systems are common among request/reply message systems. Synchronous services distribute an instant response to queries made; clients wait until the server sends a response message. The status of the service used is known to the client within a small amount of time, this is an advantage of using synchronicity in the context of the IRC chat room. Clients are actively communicating with each other and need to reply to each other’s messages in the quickest time, asynchronicity falters here with processing for requests made takes more time when compared to synchronicity. The program here adheres to this practice with the ClientHandler handling information about a client and displaying it to other clients within real time – messages from clients are sent onto the server to be viewed by other clients in the server in real time also. Responses from the ClientHandler and server being necessary to facilitate the continuous conversation between the clients connected is utilised as to ensure only a single request is processed at a time, as to prevent multiple requests slowing down the server – bottlenecking the performance, as well as the performance of the clients connected – ensuring their sockets are not overwhelmed with messages.

Interoperability was found in the language used – Java – which is an interoperable programming language, allowing itself to be written and ran on Java Virtual Machines on any device. Semantic interoperability is on display here too, with syntax errors communicated back to clients if their request had any present, allowing the message to be troubleshooted, fixed and re-delivered to the server.

TCP cannot guarantee consistency between two different endpoints, let alone multiple when several clients are connected to the server at one time, to alleviate and perhaps prevent constant

inconsistencies occurring with the packets sent – measures were taken, the ‘Two Generals Problem’ was addressed by use of the ClientHandler informing users if their message was or was not sent, or if it timed out – however there is still no way for other clients to know if a client is having connection issues, causing timeouts – other than to see said client leave the chat, with the message informing them the client has left the chat popping up if said client has left, even displaying the nickname they had too before they left – so they can be identified by other clients in the server.

The scheme the TCP uses to deliver data between a client and the server is multiplexing. Processes sent by a client are handled on the IP which the server is on before being sent out their intended destinations – those being that of the other clients connected to the server. This has advantages over the other two schemes used to deliver data with TCP such as addressing and routing. Addressing in this context would need to identify the other clients connected to the network, which would put more of an impact on the clients and rely on the connection of the multiple clients rather than a single client, routing has similar issues. Multiplexing works better in the context of an IRC chat, as IP identification of clients by other clients may lead to decreased security, and the ability to stay consistently connected cannot be confidently relied upon at all times for other clients, while relying on solely one server, which everyone is connected to can be. If there is an issue with the server itself, then it would affect everyone connected, ensuring clients have mostly the same ability to send and receive messages – increasing consistency.

Commented [AG3]: 876 words - within tolerance.

Demonstration link: <https://www.youtube.com/watch?v=2JW7I4WUyVs>