Network Distributed Multi-Client Chat Application

Pijus Dailidenas and
Cristian Dumbravanu

COMP1549: Advanced Programming
University of Greenwich
Old Royal Naval College
United Kingdom

***Abstract* – The purpose of our assignment was to create a network distributed systems that allows for multiple client to communicate with each other and with the server in a form of using a coordinator, we were using Java and our Network Programming skills to make this happen.** We chose for the project to be used through a Command Line Interface which caused some limitations; however, we were able to produce an application that allows for multiple clients to communicate with each other.

Key words: Network, Java, Multi-Client, and Communication.

I. Introduction

Following the coursework specification our objectives was to create a network distributed application that can allow for multiple clients communicate amongst each other alongside being able to communicate with the server, to achieve this we had to make sure that we understood how to use Java and how to network program amongst that we had to make sure we were able to understand other Java components such as Design Patterns, Components and JUnit Testing.

While creating this application we realized the importance of network programming, almost everything nowadays is connected to the internet in some shape or form, being able to understand how to network program is very important for the future as more applications will become reliant on the internet, It is also very important for our careers as network programming is one of the center focuses on software companies, especially with the rise of 5G and Internet of Things Applications. [1] Furthermore, there have been real world examples of network programming helping solve problems, in an article written by Fred Glover, he named multiple applications where network programming has been used to solve an issue, while the article is quite old it does show a need for network programming.

During the development of our project, we were able to create an application that abides to the specification set to us, however we ran into some issues, throughout the report we will explain our approach, design structures and the issues we had and how we can learn from them.

II. Design/implementation

Our Java multi-client communication server has utilised many design patterns that have influenced the program’s creational(how objects are created) , structural(how the objects relate to each other) and behavioural(how objects communicate with each other) aspect of development and testing. We have also followed the GRASP set of principles for assigning responsibilities of the methods and classes we used.

Our patterns follows more of a high coupling than low coupling. For example, all of the socket we have defined in the program are referenced in many methods, for example the start Server method references the server socket when connecting and listening for new clients to connect and any changes to the server socket will directly correlate with connection of clients. Other objects like the buffered writer and reader are also an example of objects that is utilised in most methods (sendMessage , listenForMessage , presentClients , ClientHandler , sendMessageTo , pingClient , broadcastMessage and removeClientHandler) this is because a Java's BufferedReader class reads text from a stream of symbols, buffering the symbols to efficiently read characters, arrays, and strings in order allow the client to read any message that was written by another client using the BufferedWrite. A problem that this has caused is that a change in an element can create unwanted changes in other aspects of the program.

Cohesion refers to how focused and related an element's responsibilities are or in other words it refers to the degree to which elements inside a module belong together. We have tried to approach and we have realised that a high cohesion class is much easier to maintain. For example, the ClinetHandler class is purely responsible for the management of the clients connecting or connected to the server. Where the clinetHandler class contains methods like ClientHandler that is responsible for assigning the coordinator and notifying other coordinates on who enters the chat and who the current coordinator. Another example would be the sendMessageTo which manages the broadcast of a private messages to specified client. And the main method being BroadcastMessage which is a centralised method that adds the majority of the functionality of the server. The creator pattern design that was utilised consists of creating instances of multiple clients which allows to run the client class multiple times making the server multiclient . It has been done with the this. command that creates many instances of the same objects. For example
:socket, bufferedWriter , bufferedReader, username and pinged.

With the idea of the information expert, we have created centralised methods that are responsible for handling any information and responsible for fulfilling the features. We have applied this to the sendMessage function in the client class where it takes the information such
assocket, bufferedReader, bufferedWriter and``username and another application of the information expert is the broadcastMessage method in the clinetHandler.

The controller (the element of the program responsible for processing system events) take form as public void broadcastMessage and the public void sendMessage which calls for other methods that each have a function that are called with a specific keyword. For example, in the client Class with send Message method allows it so if the connected client enters "exit" the sendMessage calls the closeEverything method which is responsible for closing the socket , buffered writer , buffered reader which disconnects the client. Another example is if the ClientHandler class where with the BroadcastMessage if the client enters "who" it prints out the usernames of all the clients and their ports.

We have identified and we are aware of all of the elements that if there are any changes to an element has an undesirable impact on other elements(Protected Variation). For example, if any method that are called in the controller are modified, it can make a change to the way all other methods work.

We have also applied the factory pattern that has given us a template for creating methods that are responsible for creating objects. These methods are broadcastMessage and sendMessage which all create objects for each client connected to the server

We used a Template Design Pattern which is a behaviour pattern, it was used when creating classes using the public void code, this template allowed for us to create subclasses under the template the class which allowed us to organize our code that each method we want to use is under one specific class instead of being all over the place.

We also used a Decorator Pattern which is a structural pattern, we used to by creating objects and then using those objects to add functionality to them instead of having separate classes that do the same thing all over the code.

III. Analysis and Critical Discussion

We believe that our application has been developed successfully, since multiple clients can communicate amongst each other, send private messages and communicate with the server using our application, we also used a Command Line Interface which was fully functional and responded as expected when given commands. However, we were not able to incorporate the coordinator section in detail, while we were able to inform of clients who is the coordinator and who is the first person who joins the server we were not able to incorporate a PING function to call upon other clients and get them to respond, we also were not able to assign a coordinator to a client, even though a client would be informed that they are the first to join a server and that they are the coordinator they still had all the commands available to them as the regular clients did, so when they left the server did not change coordinators, however the server was still operational and other clients could still communicate. Furthermore, to solidify our results we carried out testing of various means, we tested if the client leaves will the server work, if the client leaves will other clients still work and we found out that all of our testing hypothesis was correct and that the features we had worked no matter the conditions, the testing we carried out proved that our fault tolerance is high since we can tell that even after certain clients leave the application stays open, and even if the coordinator leaves.

We did have some limitations with our approach, since we chose to use a command line interface instead of a GUI it meant that the application was a bit compilated to use and some of the complex features like a log in page or a separate private message window was not able to be incorporated. We also were unable to incorporate a JUnit test, however we did test everything using manual methods.

Some of the improvements we could make, is to incorporate a GUI and perform JUnit tests.

IV. Conclusions

In conclusion we believe that the application was able to satisfy the goals and objective we had for it, however we do believe that one of the biggest limitations was choosing to go with a Command-Line Interface model instead of a GUI as it did not leave us much room for improving the user experience, also since our coordinator features were not fully integrated it also means we were not able to expand our application to provide more control.

Contribution:

Pijus Dailidenas – 50%
Cristian Dumbravanu – 50%

References:

https://www.cisco.com/c/en/us/solutions/enterprise-networks/what-is-network-programming.html

Glover, Fred & Klingman, Darwin. (1975). Real World Applications of Network Related Problems and Breakthroughs in Solving Them Efficiently. ACM Trans. Math. Softw.. 1. 47-55. 10.1145/355626.355634. [1]