Assignment 2 Report

COMP90015 - Distributed Systems (Semester 1, 2020)

Harfiyanto Santoso - 772503

harfiyantos@student.unimelb.edu.au

1 Introduction

The aim of this project is to design a shared whiteboards to allow multiple users to collaborate and draw concurrently on a canvas. Server-Client architecture is used in this with the help of socket to facilitate the connections between a single server and multiple clients through different threads. This report aims to describe the problem context along with the structure of the program and analysis of the work done.

2 Problem Context

There are few challenges that this implementantion aims to tackle:

1. **Concurrency:** The program needs to be able to deal with concurrency. That is, simultaneous actions by users on the shared whiteboard must lead to a reasonable state.
2. **System State:** The application needs to be structured to properly manage all the system state.
3. **Networked Communication:** Message exchange across the network needs to be managed.
4. **GUI:** The application should look similar to MS Paint.

3 Classes

This application consists of three categories of classes:

3.1 Data Type

The classes contained in this category act as a structure to hold data for certain use. There are three classes in this category:

1. **User:** Contains information from each client (e.g. name, port, socket, etc.)
2. **DrawData:** Contains information that facilitates drawing (e.g. pencil color, coordinates, text, etc.)
3. **DataArray:** ArrayList of DrawData which represents the steps taken to reach the final system state.

3.2 Server

The classes contained in this category are responsible for the Server-side GUI and functionalities. There are three classes in this category:

1. **ServerWindow:** Responsible for GUI and accepting connection requests from client(s) and creating a corresponding thread (Worker) for them.
2. **Worker:** Acts as a worker and assigned to a client to process its requests.

3. **BroadcastList:** A structure used broadcast message to all the clients by keeping an instance of BufferedWriter from each of them.

3.3 Client

The classes contained in this category are responsible for the Client-side GUI and functionalities. There are three classes in this category:

1. **ClientLogin:** Serves a login panel for the user to input their names and desired server location. It also responsible for requesting connection to the server and processing bufferedData into arraylist for PaintCanvas.
2. **ClientWindow:** This class sets up the GUI and relay any user interaction (Toolbar, Chat) to other classes.
3. **PaintCanvas:** This class renders the image on the canvas locally as well as detecting user interaction on the canvas (drawing) and inform other clients through the socket.

4 Graphical User Interface (GUI)

4.1 Server & Manager GUI *(Server.java)*

The manager who also acts as the server first need to run *Server.java* which would contain the list of active users and a kick button if the manager decides to use it. The next step would be running the same steps as users (Login GUI then Drawing GUI) therefore having two windows open at the same time.

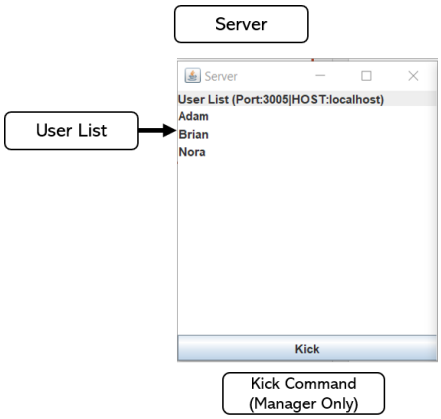

Figure 1: Server GUI

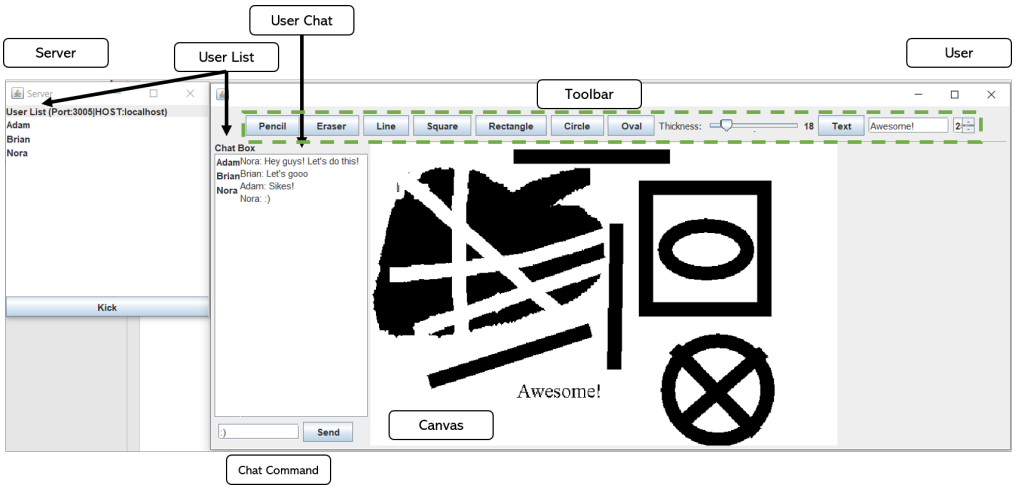

Figure 2: Manager GUI

4.2 Login GUI *(ClientLogin.java)*

The user fills the three required fields and press confirm to open the Drawing GUI.

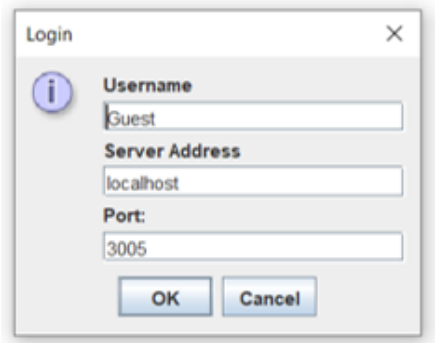

Figure 3: Login GUI

4.3 Drawing GUI*(ClientWIndow.java,PaintCanvas.java)*

This GUI is the one that normal user would see. There are five notable features in this GUI:

1. Toolbok where the user can choose from several different tools.
2. Canvas where the user can draw on.
3. Chat field and Chat command where the users can interact with each other.
4. User List where users can see other active users (WIP).
5. Auto-catchup which means that user that joins late would be able to see the latest state (Worker.java)

Although the UserList is still work in progress as the users can only see others who arrive earlier and himself.

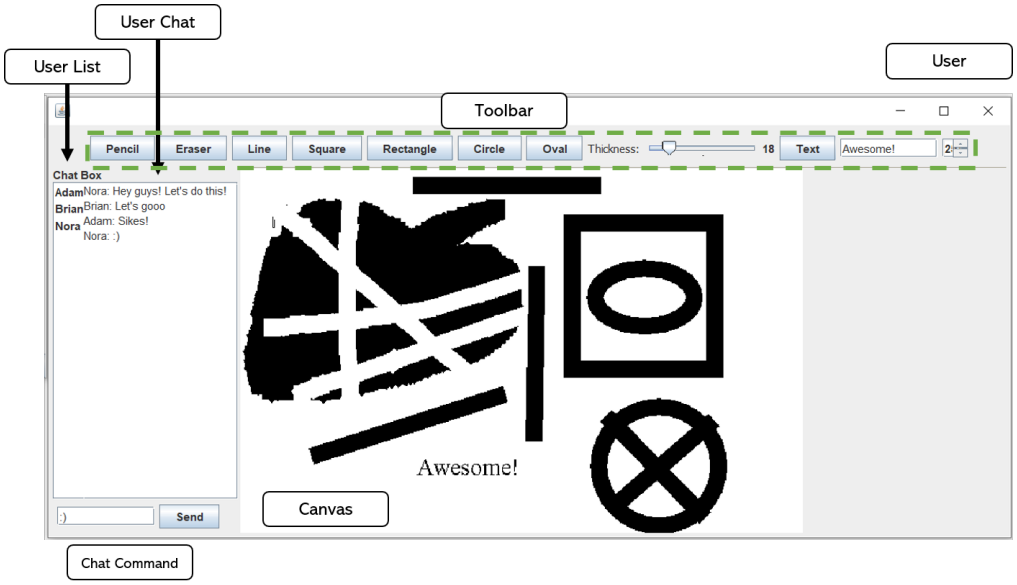

Figure 4: Drawing GUI

5 Design Analysis and Conclusion

In this implementation, the server can fulfill the requests from multiple clients concurrently. It has also meet all the problem specifications listed in Section 2.

5.1 Socket

Socket framework was chosen because of its simplicity (only a few method calls). However this may not be appropriate if this shared whiteboard was used for exchange of sensitive information as socket framework does not provide enough security (also no authentication). In that case, RMI or Servlets would more likely be appropriate as they are also fairly simple to program and have relatively better degree of security.

5.2 Thread Used

In this implementation, three threads are being used per peer:

1. First thread is used manage user logon (getting user details and providing the user with the Whiteboard's latest state (Worker.java).

2. Second thread is used send message on the chat field.

3. Finally the last and the most important thread is used to process the messages coming from different classes (Chat, Toolbox usage, New or Exiting User, and Drawing).

5.3 Propagating Modification

BroadcastList is used to broadcast the message received by a Worker thread to other threads by keeping a list of BufferedWriter for each clients. To broadcast a message to all the users, the BufferedWriter is looped through, writing the same message each time.

5.4 Possible Improvements

An obvious improvement that can be made is to create a more integrated managerial system as currently the manager has to run two different windows, the Server GUI providing access to kick command and Client GUI providing the same use (Drawing and Chatting). Additionally, the advance feature of having a file menu is also lacking in this implementation. The next step would be to integrate the three things into one simple GUI which would make it more user-friendly.

Another improvement would be to look into alternative transfer framework as discussed in previous section. While socket framework is fairly simple and sufficient for smaller contexts such as this assignment, there are better options such as RMI and Servlets that can outperform Socket framework in real world easily.