COMP90015 Distributed Systems
Assignment 2: Distributed Shared Whiteboard

Hoang Viet Mai, 813361
The University of Melbourne
May 26, 2021

1. Aim

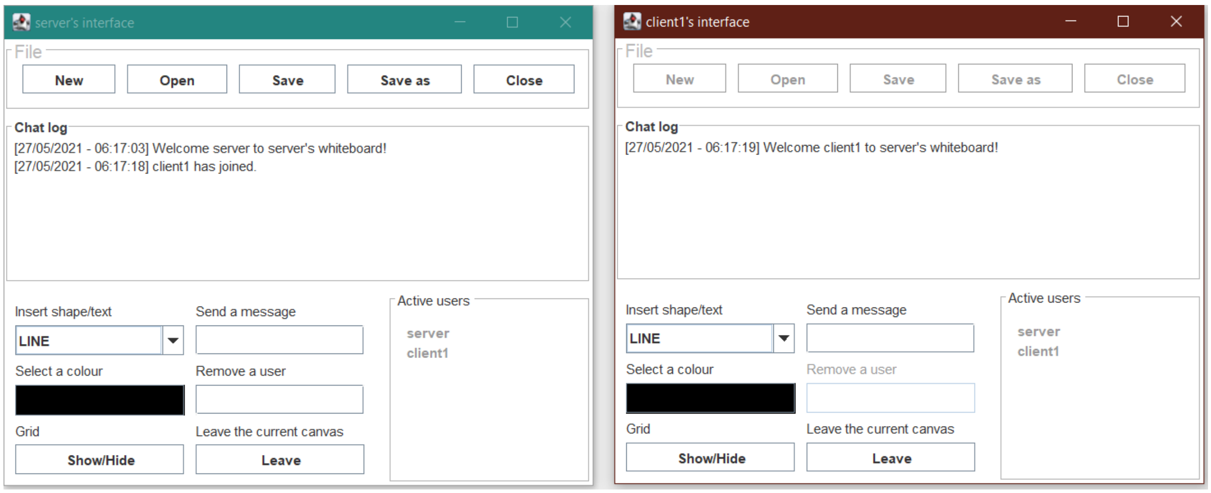
The aim of this project is to design and implement a distributed whiteboard that can be shared between multiple users over the network. The system must be implemented using Java but specific technologies are at the discretion of the developer.

Given the above context, the system was developed using Sockets with a thread-per-connection model, utilising TCP Protocol and JSON message exchange. The details of the system will be discussed in this report.

2. System architecture

2.1 Server and Client summary

Both server and client have their own whiteboard graphical user interface (GUI) for drawing operations. However, the server has exclusive administrator privileges: clear the current canvas, open a canvas from file, save the current canvas and remove a user of its choice. These privileges are reflected in the GUIs – the disabled GUI components are greyed out, which are shown in *Figure 2.1*.
The user who creates a whiteboard will start-up a whiteboard server and automatically become its manager. The server will stay alive until the manager either exits the program, leaves or closes GUI. Any incoming client must wait for the manager's approval to join the whiteboard, after which a server handler thread will be assigned to the client and their own GUI will start, with the latest whiteboard data such as drawings, texts, and chat log.
In addition to handling clients' request, the server also takes care of chat message broadcasting and canvas modification propagation between users. The basic modelling of main system components is as shown in *Figure 2.2.1*.



[27/05/2021 - 06:17:03] Welcome server to server's whiteboard!
[27/05/2021 - 06:17:18] client1 has joined.

[27/05/2021 - 06:17:19] Welcome client1 to server's whiteboard!

Figure 2.1 – Graphical user interface preview

2.2 Class diagram

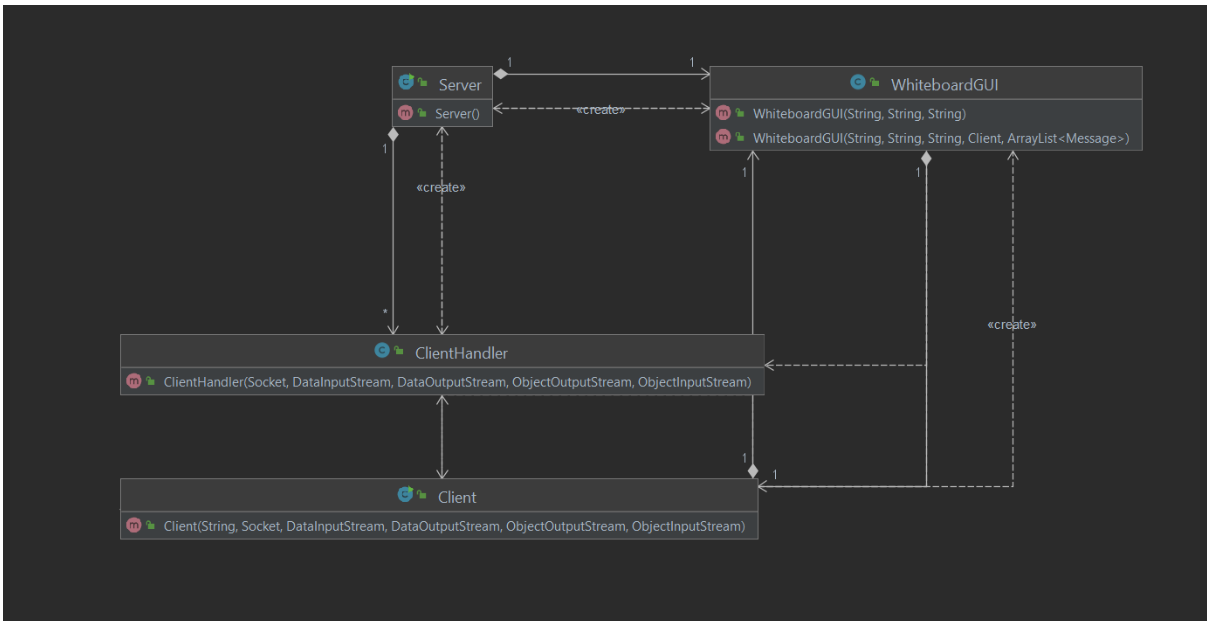

Server
Server()
WhiteboardGUI
WhiteboardGUI(String, String, String)
WhiteboardGUI(String, String, String, Client, ArrayList<Message>)
ClientHandler
ClientHandler(Socket, DataInputStream, DataOutputStream, ObjectOutputStream, ObjectInputStream)
Client
Client(String, Socket, DataInputStream, DataOutputStream, ObjectOutputStream, ObjectInputStream)
«create»
«create»
«create»
1
1
1
1
1
1
1

Figure 2.2.1 – Class diagram

2.3 Interaction diagrams (Partial)

Figure 2.3.1 and 2.3.2 shows how system components interact under the hood, as well as the order of such interactions, to satisfy the specified requirements. Only the most important objects and interactions are included in these figures due to space limitation.

2.4 Communication protocol

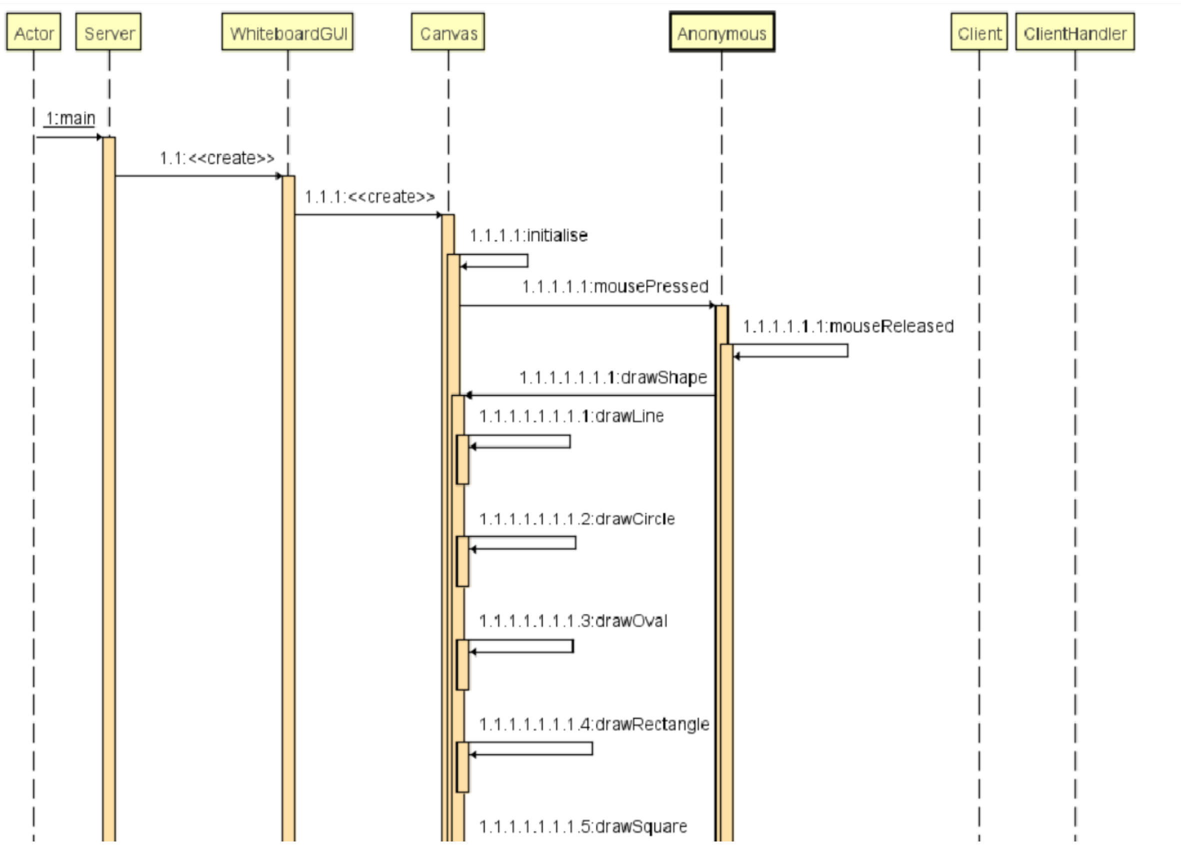


Figure 2.3.1 – Interaction diagram 1

To ensure the reliability of the connection between peers, TCP is the most suitable communication protocol for the system. However, as peers need to be able to work on the whiteboard together in real time without experiencing noticeable delays, the system requires an efficient method of propagating the modification of data from one user to the others. As the reliability provided by Java TCP Sockets are sufficient, data modification propagations are handled remotely and locally by both the Server and the Client.

Whenever a user modifies the canvas such as drawing a new shape, the data for that user will first be updated *locally* by their own whiteboard. Afterwards, a message containing the details of the action will be sent to the server. The server will parse the message to construct the new object and add it to the shared resource, after which it will broadcast the original message to all other clients one by one. Each client will have to parse the message to update their data. The same procedure applies if it was the server that makes a modification.

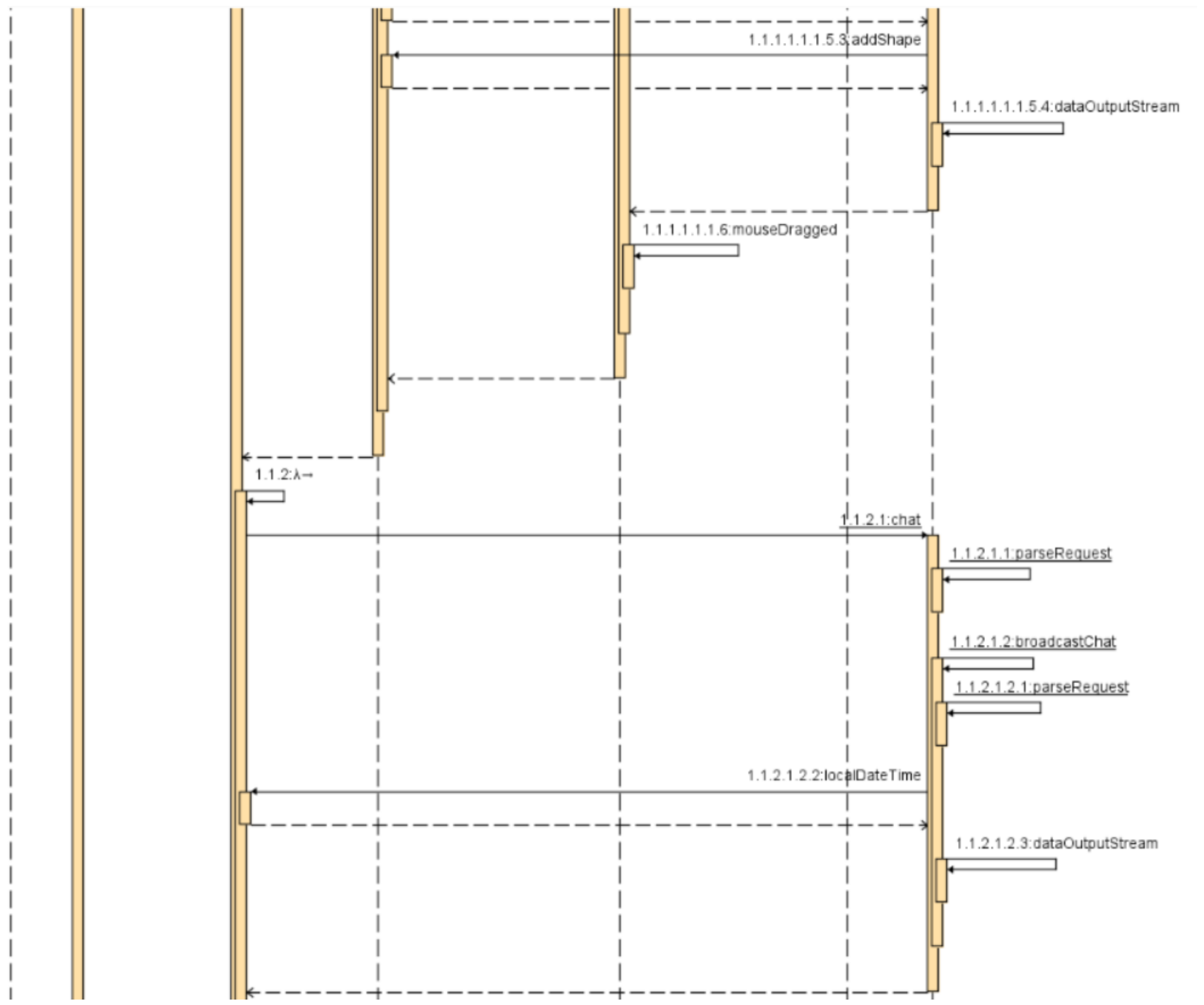


Figure 2.3.2 – Interaction diagram 2

2.5 Message exchange system

All communications between peers are in a form of UTF-8 encoded JSON Strings, which are well supported by Java socket's writeUTF method. The receiver of such messages is responsible for parsing them to extract the necessary data. This is done using the Jackson library, which supports the serialisation and deserialisation of Java Object to/from a JSON String, as long as they share the same schema. The attributes of a given message, as well as their applicability are detailed in figure 2.5.1 below.

Key	**Property**
operation	A string representing an operation. This string corresponds to a Enum constant of the same name to represent the type of request/response. Required in all operations.

user	The user who sent the message. Required in all operations.
message	Varies. Usually contains the username of the user who is the target of an operation.
text	Only used in insert text operation. Contains the string of the text object that was inserted on canvas by a user. Only used in an insert text operation.
x1	The x-coordinate of the inserted text OR the x-coordinate of the start point of a shape. Required in all drawing and insert text operations.
y1	The y-coordinate of the inserted text OR The y-coordinate of the start point of a shape. Required in all drawing and insert text operations.
colour	The colour of the text or shape. Required in all drawing and insert text operations.
shape	The type of shape that was drawn. Required in all drawing operations.
x2	The x-coordinate of the end point of a shape. Required in all drawing operations.
y2	The y-coordinate of the end point of a shape. Required in all drawing operations.

Figure 2.5.1 – Message format

3. Implementation details

3.1 Basic requirements

The following basic requirements are implemented.

Interactive canvas where users can collaborate in real time. The canvas is in responsible for all drawing operations and sending messages containing drawing details from peer to peer. Any modification to the canvas will be converted to a JSON message to be sent to the servers and/or from the server to all other users, which will then be parsed into the new added object by their respective canvas component.

The whiteboard GUI supports the drawing of various shapes: line, circle, oval, rectangle and square. The users can draw these shapes using their mouse, whose cursor's interaction with the canvas will determine the details necessary to construct the shape such as location and size. The canvas achieves this by implementing mouse action listeners. The users can also insert text at the location of their cursor.

All drawing operations are available in all colours supported by the Swing library's JColorChooser component. The current colour will be displayed in the GUI.

3.2 Advanced features

Collaborators can communicate with each other in a text-based chat system in the GUI. Chat messages are also exchanged in form of JSON strings. All users can see the chat messages sent over the life of the whiteboard, including the messages sent before they joined. Active users will have their usernames (which are enforced to be unique) displayed in real time on the GUI so that they can communicate and collaborate effectively.

Users will also be notified by a chat message whenever a user joins or leaves the whiteboard.

A manager-exclusive *File* menu with the following options:

- **New**: open a blank canvas. The server will clear their canvas by wiping all the drawings and texts (which are stored in two ArrayList objects), then send a command to all clients, instructing them to clear their whiteboard data locally. This interaction is enforced automatically when the clients receive the command.
- **Open**: loading canvas data from a saved canvas file. The update their canvas with data extracted from the file, then sent them to all users. This action can be done in the GUI, using a JFileChooser component.
- **Save** and **Save as**: save the current canvas data to a file. The former will save the data to a file with the current name in the same working directory, whereas the latter allows the managers to choose both the name and the directory of the file.
- **Close**: close the whiteboard server. All clients will receive a notification of closure, after which they will exit the program automatically.

The manager can remove a user of their choice, using the GUI. The removed user will receive a notification and forced to exit the program, by a leave command from the server. The removal of a user is recorded in the chat log.