COMP90015: Distributed Systems – Assignment 2

Distributed Shared White Board

Kaixun Yang, 1040203

The University of Melbourne

May 22, 2021

1. Problem Context

In this project, a shared whiteboard that allows multiple users to draw at the same time needs to be designed and implemented. Multiple users can use shapes and text inputting to draw on the canvas at the same time, and can choose one of 16 colors. In addition, users can also communicate with each other by typing a text through the chat window. The manager can manage the canvas (new, open, save, saveAs and close) and can authorize users to connect and kick out certain users. A Graphical User Interface (GUI) is required for this project.

2. System Design

2.1 System Architectures

The whiteboard system uses the client-server architecture which enables one server to serve multiple clients. The server is the manager and clients are users. The project uses both sockets and Java RMI.

In Java RMI, a method to request for connecting is provided, and the client can obtain the permission of the server through remotely calling the method.

In sockets, the server handles the multi-threaded clients by thread-per-connection, the thread is used to transmit drawing messages, chat messages and command messages. When the client's drawing messages or chat messages reach the server, the server will propagate messages to all other clients. At the same time, the server will save all current drawing messages, and when a new client is successfully connected, it will pass the messages to the client to ensure that all users can see the same drawing board. The server will also disseminate instructions such as close, new and kick-out to all clients at the same time. The manager has the permission to save and open files, and can propagate the opened canvas to all users

In the project, both basic features and advanced features have been successfully implemented.

For users, they can:

- Use pencil to paint
- Use shapes (line, circle, oval and rectangle) to paint
- Use text inputting to paint
- Choose one of 16 colors as the painting color
- Communicate with other users through chat window
- Get the real-time status of the canvas

For manager, he/she can:

- Authorize the connections of users
- Get the real-time status of the canvas
- Get the list of current users
- Kick-out some users by inputting their ID
- Communicate with users through chat window
- Maintain the state of the canvas (new, open, save, saveAs and close)

2.2 Communication Protocols & Message Formats

In Java RMI, the transport protocol is JRMP (Java Remote Method Protocol) which is a Java technology-specific protocol for finding and calling remote objects.

In sockets, I choose TCP as the transport protocol which can guarantee the reliability. Use '&' to connect command parameters to form a string, and convert it into a byte stream for transmission.

Command	Message Format
Chat message	"Message"& user name& message content
Pencil message	"Pencil"& color name &start x-axis& start y-axis& end x-axis& end y-axis & user name
Shape message	shape name ("Line", "Circle", "Oval", "Rectangle") & color name & start x-axis & start y-axis & end x-axis& end y-axis & user name
Text inputting message	"Text"& color name & x-axis& y-axis& text content& user name
Close the canvas message	"Close"
Create a new canvas message	"New"
Open a canvas file message	"Open"

Table 2-1 Message Formats

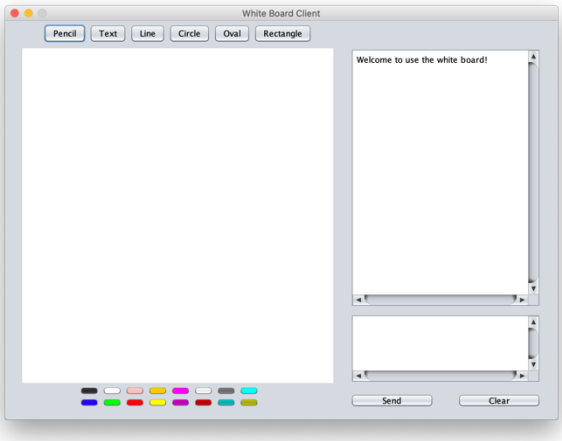

Figure 2-1 Client GUI (User)

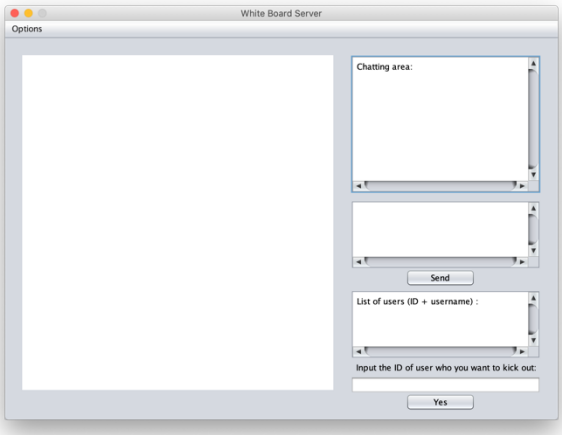

Figure 2-2 Server GUI (Manager)

3. Implementation Details

3.1 Class Design

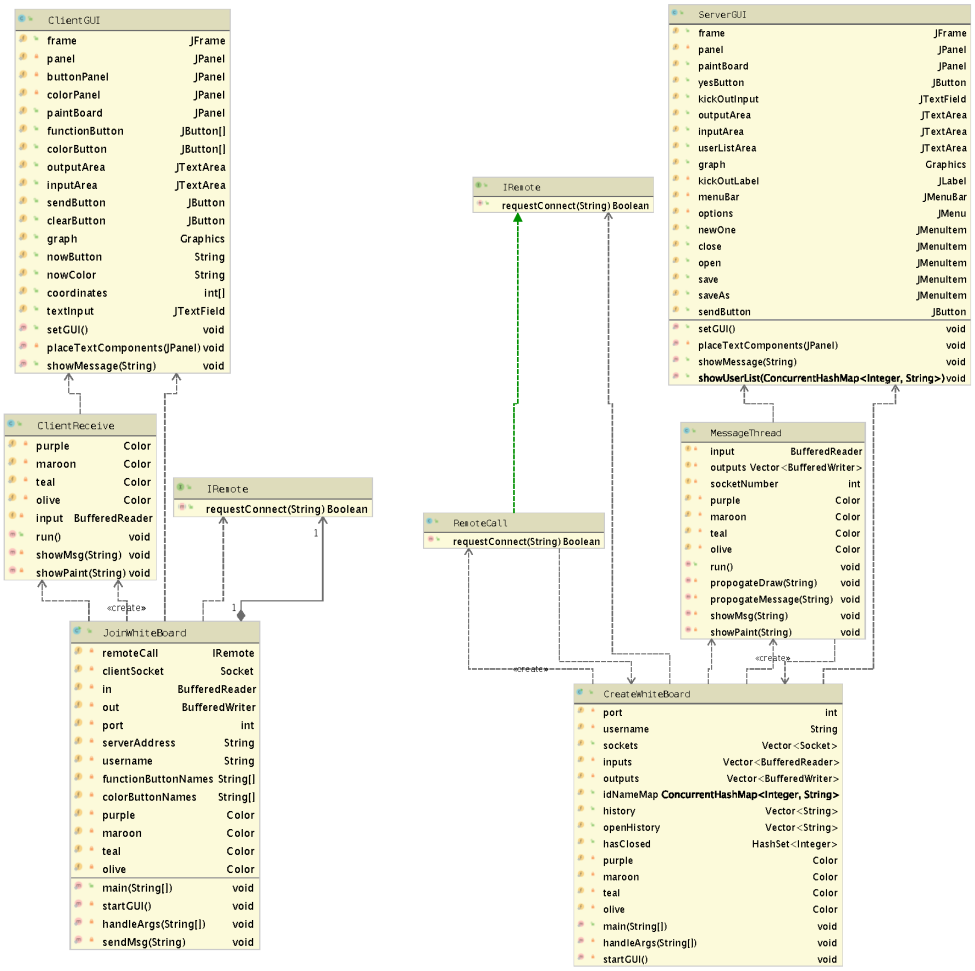

Figure 3-1 Class Diagram

There are three classes and one interface in client side:

- Interface IRmote is the interface of RMI.
- Class ClientGUI is responsible for the layout of the GUI and provides static methods to the client, including starting the GUI and displaying messages.
- Class ClientReceive inherits the thread and is used to handle the IO in the thread also provided methods to process commands to show the chat messages and paints.
- Class JoinWhiteBoard contains the main method, which is responsible for connecting to the server, binding the Java RMI, opening the client GUI, creating thread of Class ClientReceive, processing users' inputs from the GUI and sending requests to the server, calling methods in thread to showing the paints and chat messages and capturing and handling exceptions.

There are four classes and one interface in server side:

- Interface IRmote is the interface of RMI.
- Class RemoteCall is the implementation of Interface IRmote.

- Class ServerGUI is responsible for the layout of GUI and provides static methods to the server, including starting the GUI, displaying messages, showing the user list.
- Class MessageThread inherits the thread and is used to handle the IO in the thread also provided methods to process commands to show the chat messages and paints and offered the methods to send chat messages and paint messages to all other clients.
- Class CreateWhiteBoard contains the main method, which is responsible for starting the server, starting the Java RMI, opening the Server GUI, accepting clients' connections, creating thread of Class MessageThread and calling methods in thread to showing and sending the paints and chat messages, capturing and handling exceptions.

3.2 Interaction Diagram

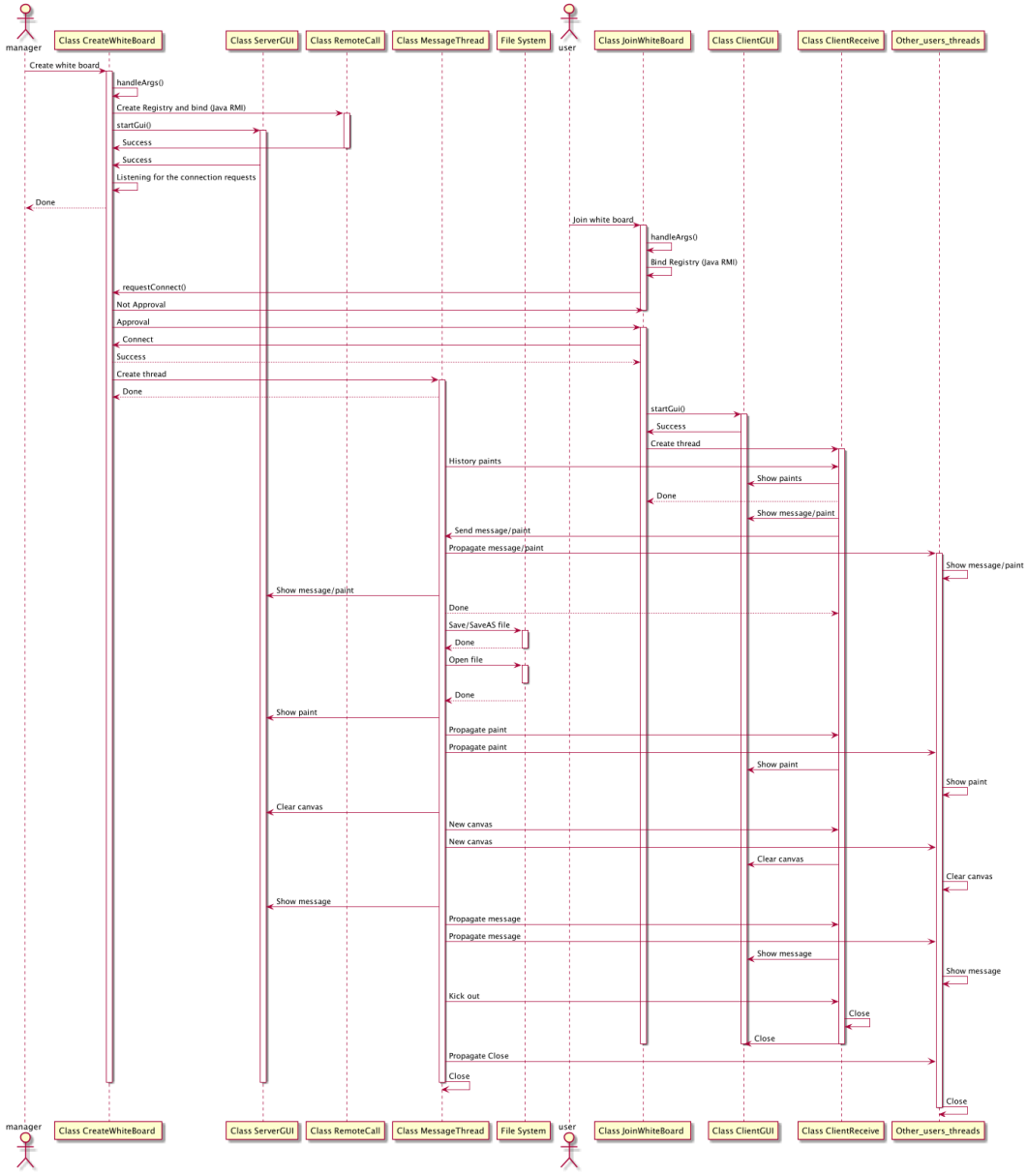

Figure 3-2 Sequence Diagram

3.3 Important Modular Design

3.3.1 Handle command line input

handleArgs() receives the parameters of the command line, which is used to check the number of parameters, the types of parameters, and whether the data range of the parameter meets expectations. If not, an exception is thrown. Both server and client have this function.

3.3.2 Handle pencil painting

Create an object of *Graphics* and assign it with *JPanel.getGraphics()*, override *mousePressed()* to get the start coordinates, and override *mouseDragged()* to get the end coordinates, then use *Graphics.drawLine()* to draw pencil line.

3.3.3 Handle shape painting

Create an object of *Graphics* and assign it with *JPanel.getGraphics()*, override *mousePressed()* to get the start coordinates, and override *mouseReleased()* get the end coordinates, then use *Graphics.drawLine(), drawOval(), drawRect()* and *drawArc()* to draw line, oval, rectangle and circle.

3.3.4 Handle text inputting painting

Create an object of *Graphics* and assign it with *JPanel.getGraphics()*,override *Graphics.mousePressed()* to get the start coordinates, and override *mouseReleased()* get the end coordinates, use *Math.min()* to get small coordinates and call *drawstring()* to draw text.

3.3.5 Handle choosing color

Use the 12 default colors of class *Color* and 4 *new Color(R, G, B)* to get total 16 colors, override *actionPerformed()* and call *Graphics.setColor()* to choose color.

3.3.6 Handle concurrency

Use key word *synchronized* to decorate the methods *propogateDraw()* and *propogateMessage()* to ensure thread safe. At the same time, use *Vector()* instead of *ArrayList()*, use *ConcurrentHashMap()* instead of *HashMap()* to ensure the concurrency.

3.3.7 Handle propagating message/paints

Use vector to store all the sockets, when server receives a message/paint, it will write the message/paint to all *OutputStream* of sockets (expect the socket of the client sending the message/paint and closed sockets).

3.3.8 Handle kicking out users

The id of user is the index of sockets vector, call *getText()* and *Integer.parseInt()* to get the kick-out id, send a message to the user through *OutputStream* of *vector.get(id)*, then the server will close the socket.

Figure 3-3 User list and kick out input

3.3.9 Handle approving connections

The client will call the RMI method *requestConnect()* to get the Boolean result to connect or not. Server can use *JOptionPane.showConfirmDialog()* to get the result from GUI.

3.3.10 Handle file menu functions

The *new* function will make server call *Graphics.repaint()* to clear the canvas and *propogateMessage()* to all the clients, and clients will call *Graphics.repaint()* too. The *close* function will make server call *System.exit(0)* to close the Server GUI and all client sockets will close and use `JOptionPane.showMessageDialog()`and *System.exit(0)* to close the clients GUI. The *save/saveAs* will save the java Object *Vector()* which contains the history paints commands. The *open* will load the java Object *Vector()* and call *Graphics.repaint()* to clear the canvas and *propogateMessage()* to all the clients, and clients will call *Graphics.repaint(),* then server will call `showPaint()`and *propogateDraw()* for every element in *Vector()* to repaint for server and all clients.

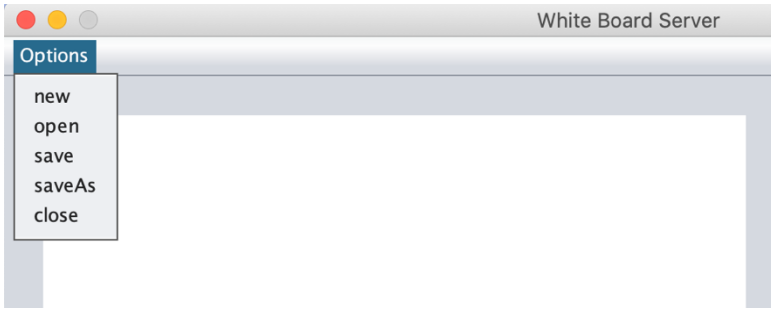

Figure 3-3 File functions

3.3.11 Handle same users' names

For each user, server will automatically assign it with an unique id, and store them in a *ConcurrentHashMap*<Int id, String name>*()*.

3.3.12 Handle user list

The server will maintain the *ConcurrentHashMap()* and show them in the GUI, if some clients are closed, the *ConcurrentHashMap()* will update too.

3.3.13 Handle new client obtaining the current state of whiteboard

The server will maintain a *vector()* to store all the history paints commands and call *sendMsg()* for every paint command to the new client, and the client will call *showPaint()* to draw them all.

4. Innovations

- Both basic features and advanced features have been successfully implemented. Additionally, a pencil tool has been implemented which allow use to draw irregular lines.
- Detailed notifications and chat message.

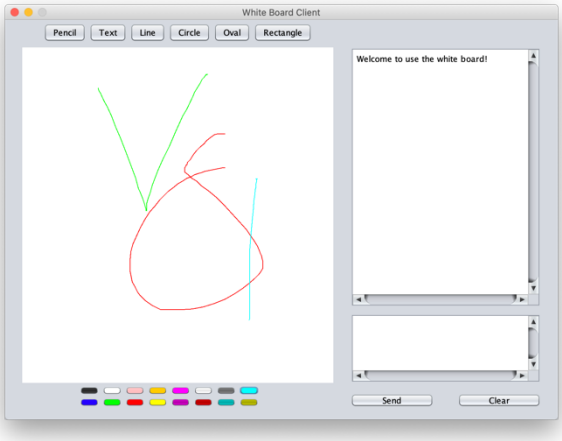

Figure 4-1 Draw irregular lines

Figure 4-2 Detailed notification for kick out

Figure 4-2 Detailed notification for closed server

Figure 4-3 Detailed chat message