Distributed Systems COMP90015
Assignment 2 Report
Distributed Shared White Board

By Rui Liu 1111181

May 30, 2022

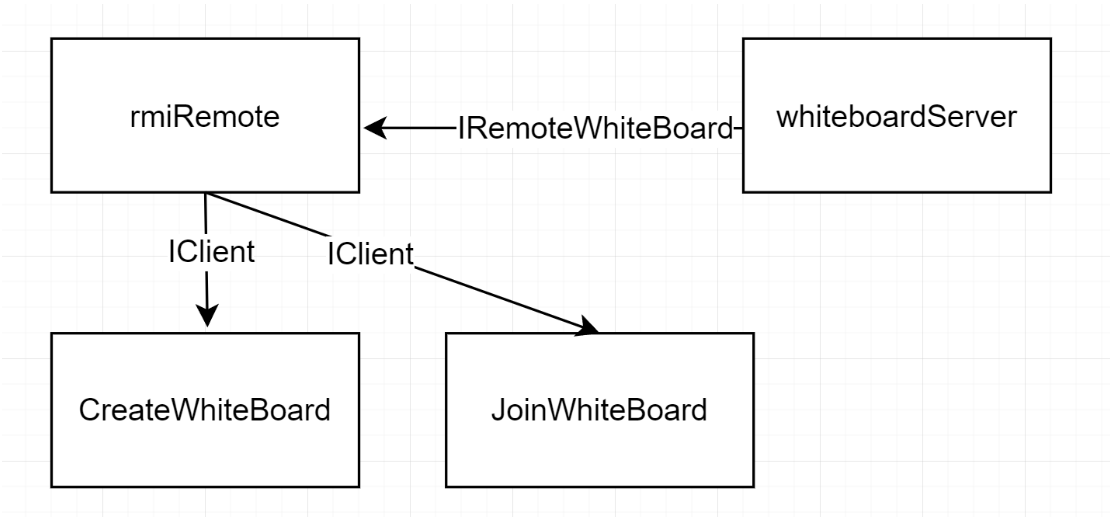
1. Introduction

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 RMI (remote method invocation), which can call the object on an object on a Java virtual machine to call another method on the object in another Java virtual machine, and it is suitable to be implement in this project.

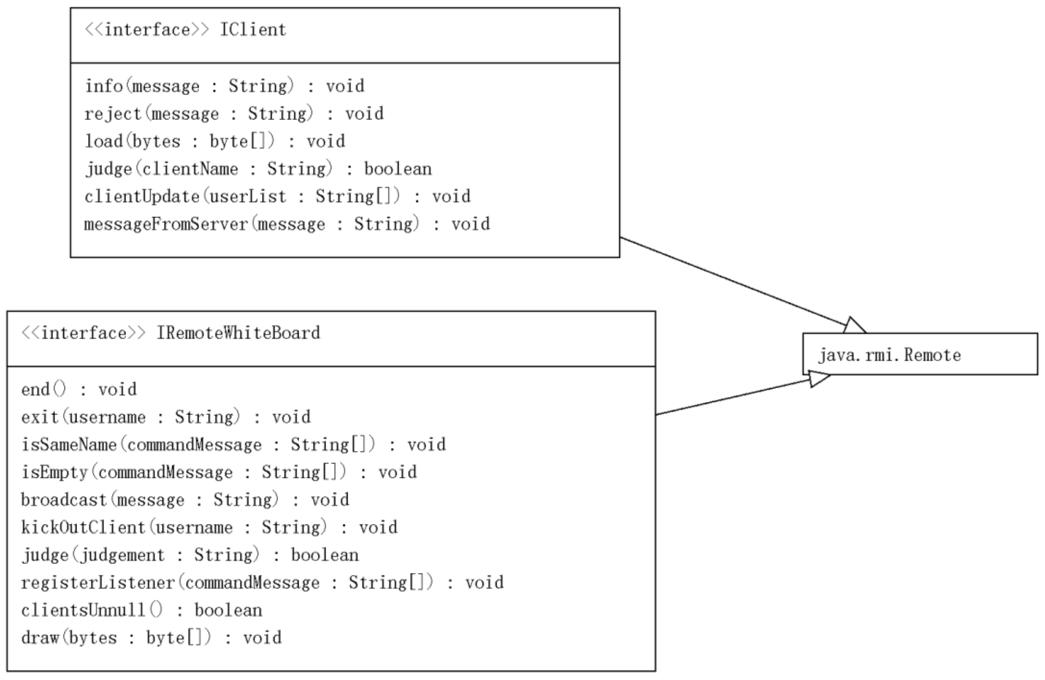
Remote method invocation is a communication mechanism that uses remote objects to call each other to achieve communication between computers. It uses remote objects to achieve communication and supports multi-threaded services. The default communication protocol is TCP/IP and the transmission layer uses the Java remote method protocol.

2. UML Diagram

2.1 interaction

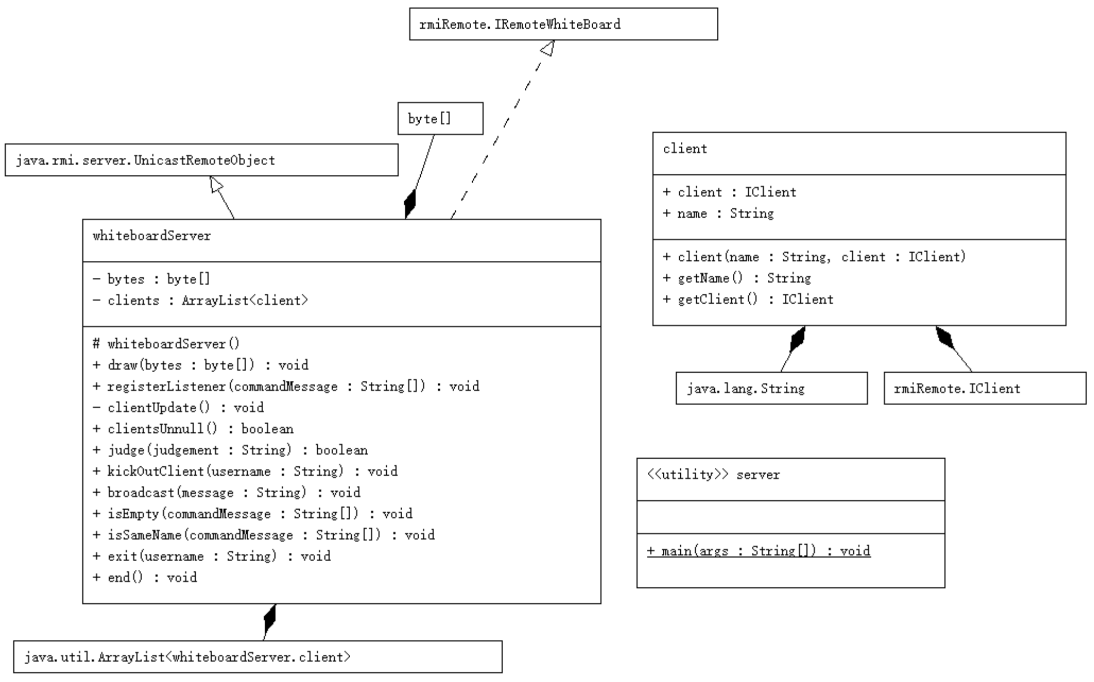


2.2 RMI Remote interface:



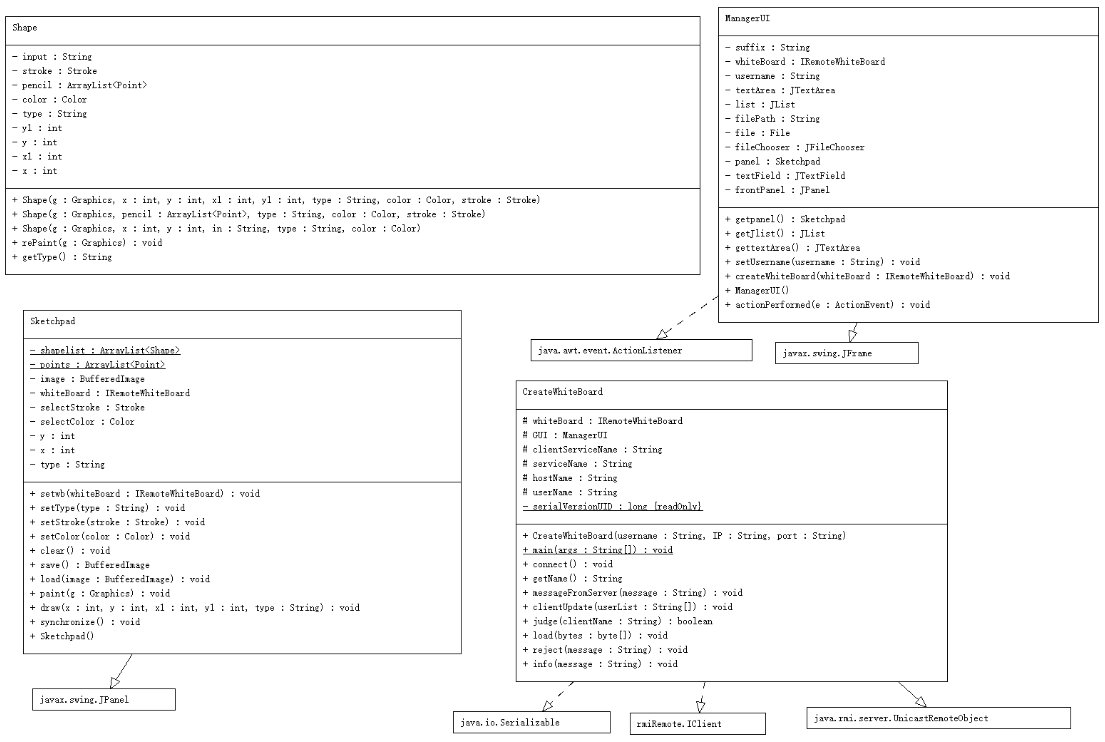
<<interface>> IClient
info(message : String) : void
reject(message : String) : void
load(bytes : byte[]) : void
judge(clientName : String) : boolean
clientUpdate(userList : String[]) : void
messageFromServer(message : String) : void
<<interface>> IRemoteWhiteBoard
end() : void
exit(username : String) : void
isSameName(commandMessage : String[]) : void
isEmpty(commandMessage : String[]) : void
broadcast(message : String) : void
kickOutClient(username : String) : void
judge(judgement : String) : boolean
registerListener(commandMessage : String[]) : void
clientsUnnull() : boolean
draw(bytes : byte[]) : void
java.rmi.Remote

2.3 Whiteboard server

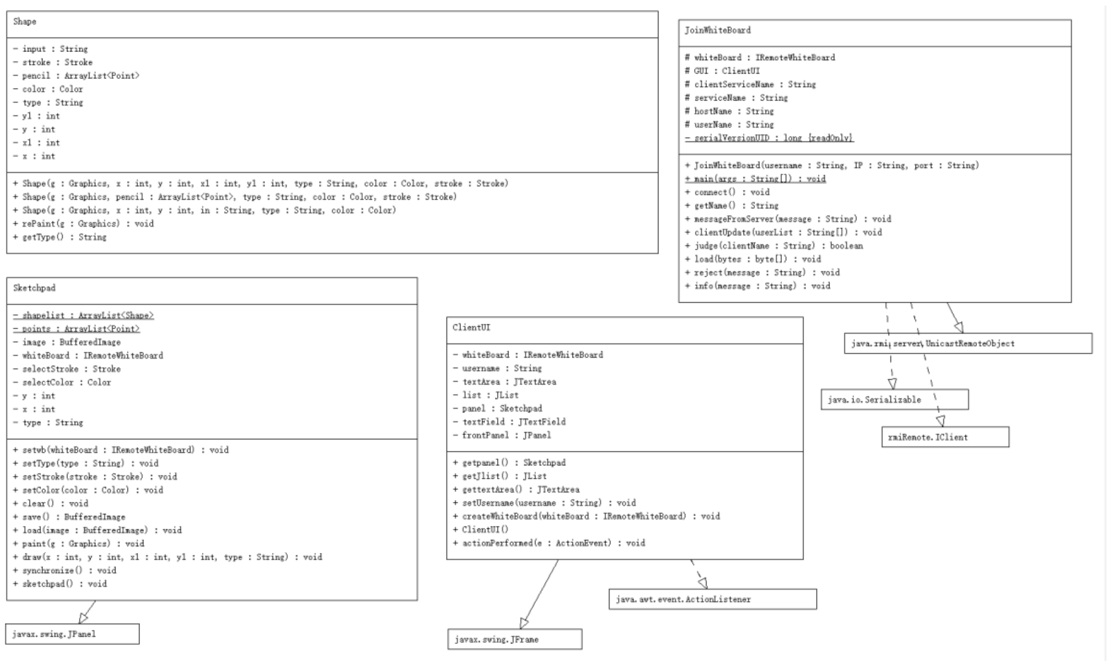


rmiRemote.IRemoteWhiteBoard
byte[]
java.rmi.server.UnicastRemoteObject
whiteboardServer
- bytes : byte[]
- clients : ArrayList<client>
# whiteboardServer()
+ draw(bytes : byte[]) : void
+ registerListener(commandMessage : String[]) : void
- clientUpdate() : void
+ clientsUnnull() : boolean
+ judge(judgement : String) : boolean
+ kickOutClient(username : String) : void
+ broadcast(message : String) : void
+ isEmpty(commandMessage : String[]) : void
+ isSameName(commandMessage : String[]) : void
+ exit(username : String) : void
+ end() : void
java.util.ArrayList<whiteboardServer.client>
client
+ client : IClient
+ name : String
+ client(name : String, client : IClient)
+ getName() : String
+ getClient() : IClient
java.lang.String
rmiRemote.IClient
<<utility>> server
+ main(args : String[]) : void

2.4 Create whiteboard by manager


Shape
- input : String
- stroke : Stroke
- pencil : ArrayList<Point>
- color : Color
- type : String
- y1 : int
- y : int
- x1 : int
- x : int
+ Shape(g : Graphics, x : int, y : int, x1 : int, y1 : int, type : String, color : Color, stroke : Stroke)
+ Shape(g : Graphics, pencil : ArrayList<Point>, type : String, color : Color, stroke : Stroke)
+ Shape(g : Graphics, x : int, y : int, in : String, type : String, color : Color)
+ rePaint(g : Graphics) : void
+ getType() : String
ManagerUI
- suffix : String
- whiteBoard : IRemoteWhiteBoard
- username : String
- textArea : JTextArea
- list : JList
- filePath : String
- file : File
- fileChooser : JFileChooser
- panel : Sketchpad
- textField : JTextField
- frontPanel : JPanel
+ getpanel() : Sketchpad
+ getJlist() : JList
+ gettextArea() : JTextArea
+ setUsername(username : String) : void
+ createWhiteBoard(whiteBoard : IRemoteWhiteBoard) : void
+ ManagerUI()
+ actionPerformed(e : ActionEvent) : void
java.awt.event.ActionListener
javax.swing.JFrame
Sketchpad
- shapelist : ArrayList<Shape>
- points : ArrayList<Point>
- image : BufferedImage
- whiteBoard : IRemoteWhiteBoard
- selectStroke : Stroke
- selectColor : Color
- y : int
- x : int
- type : String
+ setwb(whiteBoard : IRemoteWhiteBoard) : void
+ setType(type : String) : void
+ setStroke(stroke : Stroke) : void
+ setColor(color : Color) : void
+ clear() : void
+ save() : BufferedImage
+ load(image : BufferedImage) : void
+ paint(g : Graphics) : void
+ draw(x : int, y : int, x1 : int, y1 : int, type : String) : void
+ synchronize() : void
+ Sketchpad()
javax.swing.JPanel
CreateWhiteBoard
# whiteBoard : IRemoteWhiteBoard
# GUI : ManagerUI
# clientServiceName : String
# serviceName : String
# hostName : String
# userName : String
- serialVersionUID : long {readOnly}
+ CreateWhiteBoard(username : String, IP : String, port : String)
+ main(args : String[]) : void
+ connect() : void
+ getName() : String
+ messageFromServer(message : String) : void
+ clientUpdate(userList : String[]) : void
+ judge(clientName : String) : boolean
+ load(bytes : byte[]) : void
+ reject(message : String) : void
+ info(message : String) : void
java.io.Serializable
rmiRemote.IClient
java.rmi.server.UnicastRemoteObject

2.5 Join whiteboard by users


Shape
- input : String
- stroke : Stroke
- pencil : ArrayList<Point>
- color : Color
- type : String
- y1 : int
- y : int
- x1 : int
- x : int
+ Shape(g : Graphics, x : int, y : int, x1 : int, y1 : int, type : String, color : Color, stroke : Stroke)
+ Shape(g : Graphics, pencil : ArrayList<Point>, type : String, color : Color, stroke : Stroke)
+ Shape(g : Graphics, x : int, y : int, in : String, type : String, color : Color)
+ rePaint(g : Graphics) : void
+ getType() : String
JoinWhiteBoard
# whiteBoard : IRemoteWhiteBoard
# GUI : ClientUI
# clientServiceName : String
# serviceName : String
# hostName : String
# userName : String
- serialVersionUID : long {readOnly}
+ JoinWhiteBoard(username : String, IP : String, port : String)
+ main(args : String[]) : void
+ connect() : void
+ getName() : String
+ messageFromServer(message : String) : void
+ clientUpdate(userList : String[]) : void
+ judge(clientName : String) : boolean
+ load(bytes : byte[]) : void
+ reject(message : String) : void
+ info(message : String) : void
java.rmi.server.UnicastRemoteObject
java.io.Serializable
rmiRemote.IClient
Sketchpad
- shapelist : ArrayList<Shape>
- points : ArrayList<Point>
- image : BufferedImage
- whiteBoard : IRemoteWhiteBoard
- selectStroke : Stroke
- selectColor : Color
- y : int
- x : int
- type : String
+ setwb(whiteBoard : IRemoteWhiteBoard) : void
+ setType(type : String) : void
+ setStroke(stroke : Stroke) : void
+ setColor(color : Color) : void
+ clear() : void
+ save() : BufferedImage
+ load(image : BufferedImage) : void
+ paint(g : Graphics) : void
+ draw(x : int, y : int, x1 : int, y1 : int, type : String) : void
+ synchronize() : void
+ sketchpad() : void
javax.swing.JPanel
ClientUI
- whiteBoard : IRemoteWhiteBoard
- username : String
- textArea : JTextArea
- list : JList
- panel : Sketchpad
- textField : JTextField
- frontPanel : JPanel
+ getpanel() : Sketchpad
+ getJlist() : JList
+ gettextArea() : JTextArea
+ setUsername(username : String) : void
+ createWhiteBoard(whiteBoard : IRemoteWhiteBoard) : void
+ ClientUI()
+ actionPerformed(e : ActionEvent) : void
java.awt.event.ActionListener
javax.swing.JFrame

3. System Components

3.1 rmiRemote

In this packet, it includes two interfaces: IClient and IRemoteWhiteBoard.

IClient contains the methods in the client and enables the server to use.

IRemoteWhiteBoard contains the methods in the server and enables the client to use.

3.2 whiteboardServer

In this packet, it includes three classes: whiteboardServer, server and client.

whiteboardServer contains a series of operation methods, such as painting, broadcasting, adding and removing new users, etc. The new user registration uses synchronize, and a new thread is opened when the user is removed to ensure concurrency problems.

Class server is used to start the server and rmi.

Class client is used to store the information of clients.

3.3 CreateWhiteBoard

In this packet, it includes four classes: CreateWhiteBoard, managerUI, sketchpad and Shape.

CreateWhiteBoard is used to create a whiteboard object and enable the manager to perform operations.

ManagerUI creates the function of GUI of the manager and has the function to open or save files, choose shape and colour to paint, sending message to others and reminding users the operator.

Sketchpad uses graphics2D to implement image painting function and synchronizes images to all users by sending pictures.

Shape uses graphics2D to implement the repaint method which can guarantee to keep the original painting function after minimization.

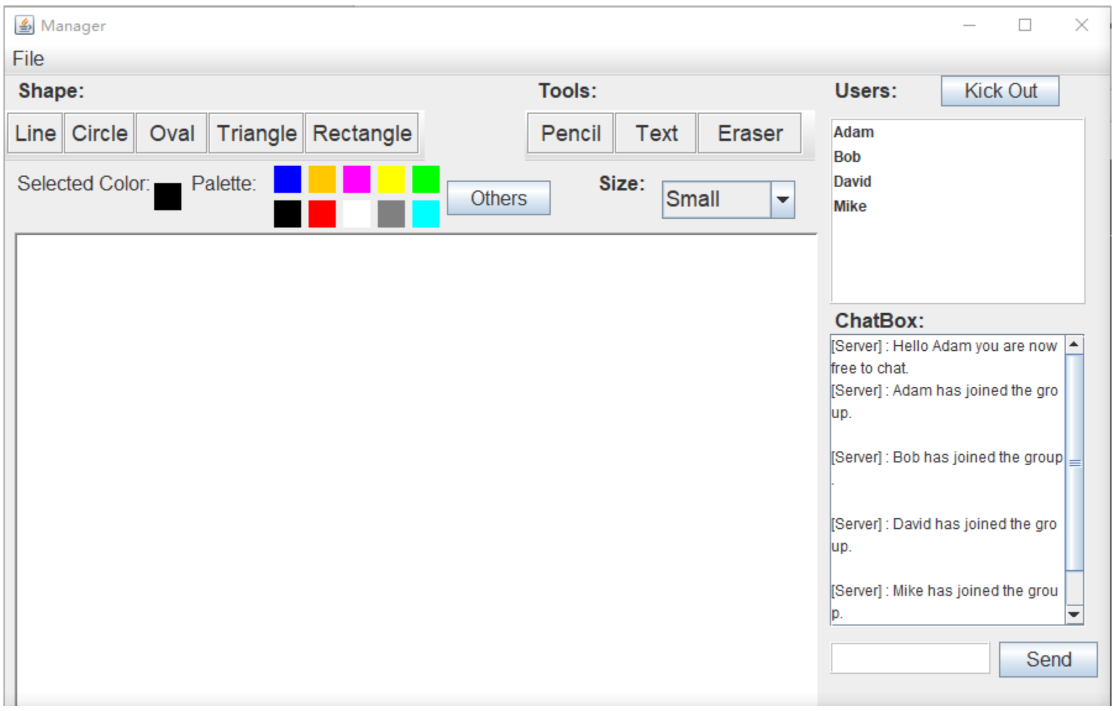
3.4 JoinWhiteBoard

In this packet, it includes four classes: JoinWhiteBoard, clientUI, sketchpad and Shape which have the most same function of CreateWhiteBoard, except manager functions to save file and kick out clients.

4. Graphical User Interface (GUI)

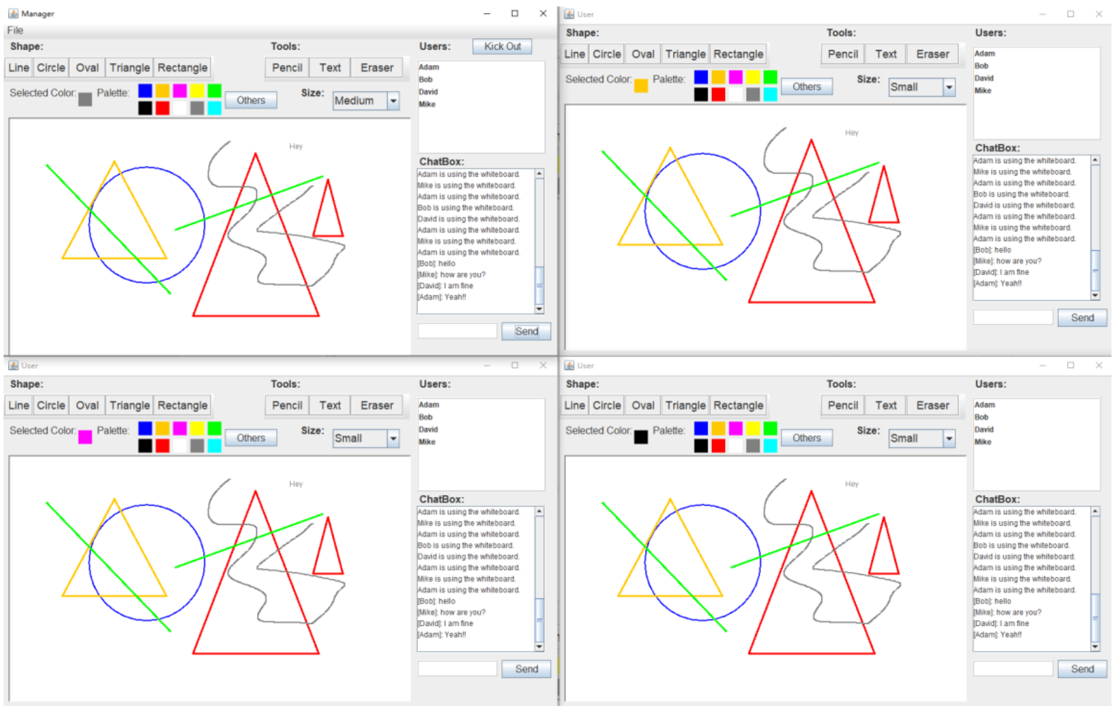
Comparison of manager GUI and client GUI: it supports the basic shape line, circle, oval, triangle and rectangle, and supports user to use pencil, text in any place, and can use eraser to wipe image with three types of size: small, medium and large.

And the chatbox shows the messages users send and broadcasts the person who are painting. The manager can kick out clients by using Kick Out button and typing the name of client.



Adam
Bob
David
Mike

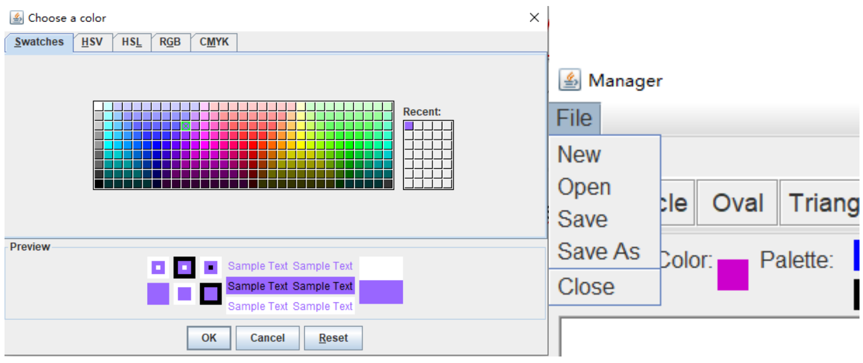
[Server] : Hello Adam you are now free to chat.
[Server] : Adam has joined the group.
[Server] : Bob has joined the group.
[Server] : David has joined the group.
[Server] : Mike has joined the group.



Adam
Bob
David
Mike

Adam is using the whiteboard.
Mike is using the whiteboard.
Adam is using the whiteboard.
Bob is using the whiteboard.
David is using the whiteboard.
Adam is using the whiteboard.
Mike is using the whiteboard.
Adam is using the whiteboard.
[Bob]: hello
[Mike]: how are you?
[David]: I am fine
[Adam]: Yeah!!

It supports basic 10 colours and can use ‘others’ to open colorchooser. And the manager can use File menu to open or save the image.



5. New Innovation

In this application, it allows users to communicate with others by typing texts, allows manager to use File menu and allows the manager to kick out a certain user.

Meanwhile, it allows clients to use pencil to paint with mouse, use text to type text in the whiteboard, use eraser to wipe image and use Size to select the size of pattern of lines.

Finally, it allows users to understand the person who are using the whiteboard.

6. Critical Analysis and Conclusion

In this application, server-Client architecture is used in this with RMI (remote method invocation). Although RMI has the advantages of object -oriented, high security, and more concise, it also has some shortcomings, such as RMI's IP address and port dependencies on the server is closely dependent. This is widely solved by DNS, or exposed IP to the program code by encapsulation.

Furthermore, RMI needs to occupy more network bandwidth. So in the future, we should choose the appropriate method for programming when appropriate, because each protocol and structure can have their advantages and disadvantages.