COMP90015: Distributed Systems – Assignment 2

Distributed Shared White Board

THE UNIVERSITY OF MELBOURNE

Group name: unimelb

Name	Student ID	Contribution area	Overall contribution to Project
Student Email			
Chenao Xie	1041989	Whiteboard communication	25%
chenao.xie@student.unimelb.edu.au			
Jiayang Ao	1046679	Chat window	25%
jiayanga@student.unimelb.edu.au			
Siyang Wang	890524	Parts of different functions	25%
siyangw4@student.unimelb.edu.au			
Xinnan Shen	1051380	Drawing board	25%
xinnan.shen@student.unimelb.edu.au			

1. Introduction

Whiteboard has always been the preferred way of drawing, teaching and demonstration. With the development of distributed systems and computer science, distributed shared whiteboard breaks the physical limitations of the traditional whiteboard and provides collaborative and flexible attributes to share drawings and ideas. When people work with their team members, they can add comments, draw images, or extract resources directly from the shared whiteboard anytime and anywhere. In addition, the whiteboard allows people to unleash the potential of the team through real-time co-creation.

This report describes our group assignment of designing and implementing distributed shared whiteboard implemented in Java, which allows multiple clients over the network to perform a series of simultaneous actions in the graphical user interface (GUI), including：

- Choose the colour.
- Free draw with the mouse.
- Insert lines that can be changed the length and moved.
- Insert different shapes (oval, circular, square, rectangular) that can be resized and moved.
- Insert text using different fonts.
- Use an eraser of five sizes.
- New, open, save, save As and close.

Besides, this whiteboard also provides a text-based chat window that allows all current users of the system to broadcast messages in the whole group or chat in private. Users are able to choose emoticons or use the translation function in the input box when chatting.

2. System Components

2.1 System architecture

This system uses a client-server architecture. The connection between clients and server is based on the RMI technique and the socket using TCP protocol so that the reliability of communication can be guaranteed. The method of the thread pool is used to implement the multi-threaded server and clients. To prevent the inconsistent state, we set the methods of handling the data to be synchronized. Having done that, multiple clients can access the server concurrently, and they do not need to worry about the inconsistent state.

As for the transmission of the data, the chat window uses the JSON format to transmit the data, while the whiteboard use serialization object to transmit the data between the server and clients.

In terms of the design of GUI in both client and server, the system uses the WindowBuilder plugin to build the GUI in Java Swing framework so that the users can use the system much more conveniently. WindowBuilder is a useful plugin to help us to implement the GUI much more efficiently than purely coding to build GUI in Java Swing Framework.

2.2 Communication protocols

There are many types of development techniques based on client-server (CS) architecture, such as Socket, Java RMI, CORBA, etc. In order to better understand these techniques, we will discuss the advantages and disadvantages of RMI and TCP in this chapter. Finally, we will propose an integrated framework of Java Socket (TCP) and RMI and apply it to our whiteboard implementation.

2.2.1 RMI

RMI enables flexible and extensible distributed communication between Java programs. It allows Java objects to exist in multiple different address spaces, distributed across different Java virtual machines. Each address space can be on the same host or on different computers within the network. Because remote method calls span different virtual machine boundaries to different specified address spaces, there are no global variables shared by objects, which requires object serialization API. This API enables Java objects to be passed between different JVM. Object serialization is specifically designed for Java objects, which means that objects in Java programs can be accessed by using object parameters. Combined with RMI and object serialization mechanisms, developers can access objects and data that cross the boundaries of the local Java virtual machine. Remote methods of remote objects can be called through RMI, and objects can be passed to these methods through the Java object serialization mechanism.

RMI provides a simple and direct model for distributed computing on the Java platform. As the Java RMI is based on the Java platform, it brings the security and portability of the Java platform to distributed computing.

All in all, Java embodies the idea that "once you write a program, you can run it anywhere." Whereas RMI extends Java pattern to make this idea a reality.

2.2.2 TCP

TCP is a connection-oriented, reliable, byte stream-based transport layer communication protocol. Java provides two classes for the TCP protocol: socket class and ServerSocket class. A Socket instance represents a client of a TCP connection, while a ServerSocket instance represents a server of a TCP connection. In TCP Socket programming, there can be multiple clients but only one server. The client sends a connection request to the server, and the server-side ServerSocket instance listens for TCP connection requests from the client and creates a new Socket instance for each request. Since the server blocks when calling accept () to wait for the client's connection request and will not continue to execute code until the connection request sent by the client is received, a thread needs to be opened for each Socket connection. The server needs to processes both the ServerSocket instance and the Socket instance, while the client only needs to use the Socket instance. In addition, each Socket instance is associated with an InputStream and OutputStream object, and we send the data by writing bytes to the socket's OutputStream and receiving the data from the InputStream.

2.2.3 An integrated framework of TCP and RMI

RMI and TCP have their own advantages and disadvantages. First, RMI is object-oriented, while Sockets is not. Second, RMI is bound to the language. For example, when using Java RMI, both client and server must be implemented in Java. While Sockets is independent of the development language and even independent of the platform. Third, RMI and Sockets are at different levels from the point of view of the network protocol stack. RMI defines its own application protocol based on TCP protocol, and its transport layer adopts Java remote method protocol (JRMP). As a result, RMI loses some flexibility and control compared to Sockets, but the advantage is that it gives developers more simplicity and convenience. Last but not least, the performance of the two methods is different, that often determines which technology developers will use to develop applications. Compared with TCP based socket, RMI needs to consume more protocol overhead to transmit the same valid data. When an application does not need to transfer a lot of data between client and server, RMI is a good choice because it is simple and easy to develop. However, once an application needs to transfer a lot of data between client and server, we should use Sockets instead of RMI.

In practice, we used TCP in the chat window, used RMI in the canvas, and combined them together in a framework to apply to our whiteboard implementation. The combination of Socket and RMI can effectively improve the scalability and execution performance of network applications.

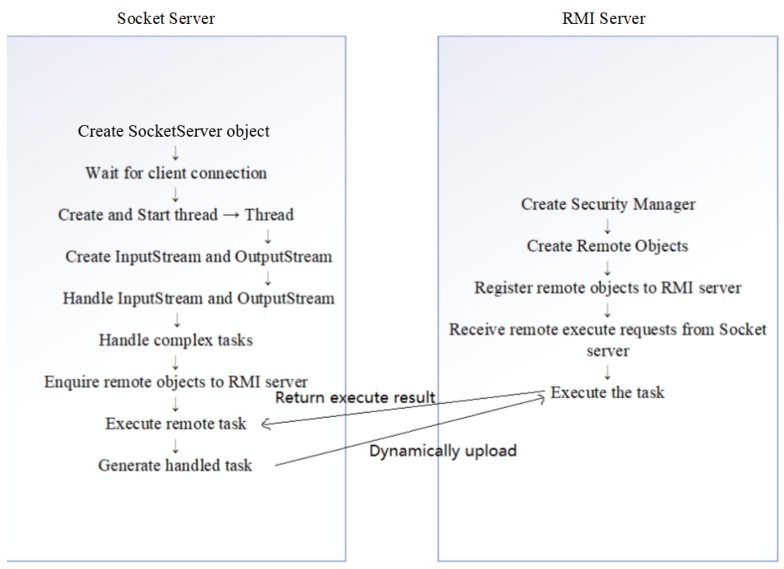
Socket Server
RMI Server
Create SocketServer object
Wait for client connection
Create and Start thread → Thread
Create InputStream and OutputStream
Handle InputStream and OutputStream
Handle complex tasks
Enquire remote objects to RMI server
Execute remote task
Generate handled task
Create Security Manager
Create Remote Objects
Register remote objects to RMI server
Receive remote execute requests from Socket server
Execute the task
Return execute result
Dynamically upload

Figure 1: Integrated framework of TCP and RMI

RMI technology enables Java programs on one computer to execute objects remotely on another computer. The result of execution can be returned to the Java program through the parameters of the method. It involves two aspects of the program: the server and the client. The server creates a series of objects, registers with the RMI registration server, and waits for the client to execute. The client gets a reference to the object on the server and calls its method remotely. These methods are executed on the machine where the server program is located, and the parameters of the method are returned to the client. After the integration of Java Socket program and RMI, the complex processing tasks can be handed over to the RMI server. At the same time, the dynamic code calling mechanism of Java can be used to realize the dynamic expansion of the code, which makes the server more scalable and extensible.

2.3 Message formats

JSON is a lightweight open-standard file format with filenames use the extension .json. We choose JSON as our data exchange format because it is convenient for transmission and conversion, less redundant characters and easy to read.

JSON's basic data types are Number, String, Boolean, Array, Object and null. In this project, we will use JSON as the data format to store and transmit data between server and clients. The reason for using JSON is to ensure the reliability of the system and try to avoid the possible delay between clients and servers. The delay is likely to happen when we transmit a large number of data, so by using JSON we can minimize the possibility of the delay.

3. System Design and Implementation details

3.1 Class Design

The detailed class designing structure is shown in the graph below.

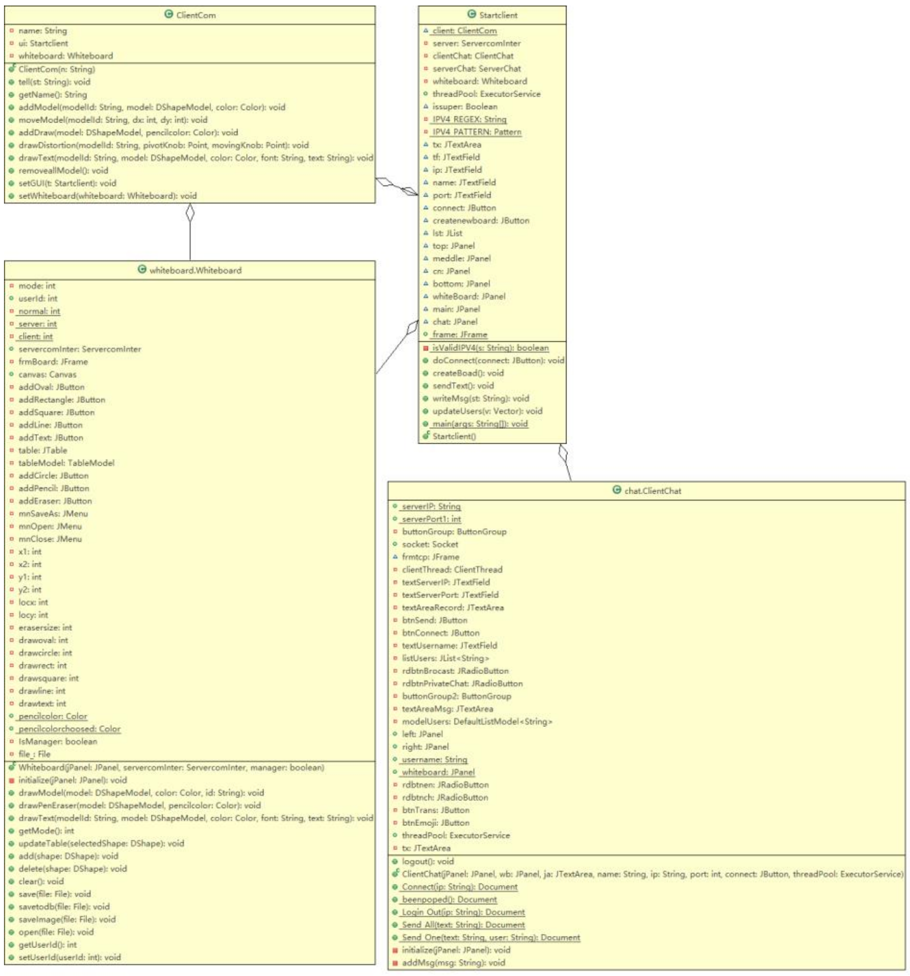
ClientCom
name: String
ui: Startclient
whiteboard: Whiteboard
ClientCom(n: String)
tell(st: String): void
getName(): String
addModel(modelId: String, model: DShapeModel, color: Color): void
moveModel(modelId: String, dx: int, dy: int): void
addDraw(model: DShapeModel, pencilcolor: Color): void
drawDistortion(modelId: String, pivotKnob: Point, movingKnob: Point): void
drawText(modelId: String, model: DShapeModel, color: Color, font: String, text: String): void
removeallModel(): void
setGUI(t: Startclient): void
setWhiteboard(whiteboard: Whiteboard): void
Startclient
client: ClientCom
server: ServercomInter
clientChat: ClientChat
serverChat: ServerChat
whiteboard: Whiteboard
threadPool: ExecutorService
issuper: Boolean
IPV4 REGEX: String
IPV4 PATTERN: Pattern
tx: JTextArea
tf: JTextField
ip: JTextField
name: JTextField
port: JTextField
connect: JButton
createnewboard: JButton
lst: JList
top: JPanel
meddle: JPanel
cn: JPanel
bottom: JPanel
whiteBoard: JPanel
main: JPanel
chat: JPanel
frame: JFrame
isValidIPV4(s: String): boolean
doConnect(connect: JButton): void
createBoad(): void
sendText(): void
writeMsg(st: String): void
updateUsers(v: Vector): void
main(args: String[]): void
Startclient()
whiteboard.Whiteboard
mode: int
userId: int
normal: int
server: int
client: int
servercomInter: ServercomInter
frmBoard: JFrame
canvas: Canvas
addOval: JButton
addRectangle: JButton
addSquare: JButton
addLine: JButton
addText: JButton
table: JTable
tableModel: TableModel
addCircle: JButton
addPencil: JButton
addEraser: JButton
mnSaveAs: JMenu
mnOpen: JMenu
mnClose: JMenu
x1: int
x2: int
y1: int
y2: int
locx: int
locy: int
erasersize: int
drawoval: int
drawcircle: int
drawrect: int
drawsquare: int
drawline: int
drawtext: int
pencilcolor: Color
pencilcolorchoosed: Color
IsManager: boolean
file_: File
Whiteboard(jPanel: JPanel, servercomInter: ServercomInter, manager: boolean)
initialize(jPanel: JPanel): void
drawModel(model: DShapeModel, color: Color, id: String): void
drawPenEraser(model: DShapeModel, pencilcolor: Color): void
drawText(modelId: String, model: DShapeModel, color: Color, font: String, text: String): void
getMode(): int
updateTable(selectedShape: DShape): void
add(shape: DShape): void
delete(shape: DShape): void
clear(): void
save(file: File): void
savetodb(file: File): void
saveImage(file: File): void
open(file: File): void
getUserId(): int
setUserId(userId: int): void
chat.ClientChat
serverIP: String
serverPort1: int
buttonGroup: ButtonGroup
socket: Socket
frmtcp: JFrame
clientThread: ClientThread
textServerIP: JTextField
textServerPort: JTextField
textAreaRecord: JTextArea
btnSend: JButton
btnConnect: JButton
textUsername: JTextField
listUsers: JList<String>
rdbtnBrocast: JRadioButton
rdbtnPrivateChat: JRadioButton
buttonGroup2: ButtonGroup
textAreaMsg: JTextArea
modelUsers: DefaultListModel<String>
left: JPanel
right: JPanel
username: String
whiteboard: JPanel
rdbtnen: JRadioButton
rdbtnch: JRadioButton
btnTrans: JButton
btnEmoji: JButton
threadPool: ExecutorService
tx: JTextArea
logout(): void
ClientChat(jPanel: JPanel, wb: JPanel, ja: JTextArea, name: String, ip: String, port: int, connect: JButton, threadPool: ExecutorService)
Connect(ip: String): Document
beenpoped(): Document
Login Out(ip: String): Document
Send All(text: String): Document
Send One(text: String, user: String): Document
initialize(jPanel: JPanel): void
addMsg(msg: String): void

Figure 2: Classes Structure

There are multiple classes in both clients and the server. Both the clients and the server are composed of three main packages, namely chat, RMI and whiteboard. These packages are used for chat functions, RMI implementation and drawing functions respectively. There are many classes in each package and they serve for very different functions. In the chat package, there are two main classes. One of them is used as chat server, while the other one is used as chat client. By implementing both classes, the chat can come to reality. Regarding RMI package, there are two classes as well. One of them acts as RMI server while the other one acts as RMI client. However, the class structure is quite complicated in whiteboard package, and there are plenty of classes in the package. To be specific, there are two classes for each shape. One is used to store data regarding the shape while the other is used to store data regarding the specific shape model. Apart from that, there are two extra classes to handle the overall whiteboard data storage and the methods of the whiteboard.

3.2 Interaction design

In this project, we have utilized TCP and RMI to interact between server and clients. In the part of the drawing board, RMI is used to transmit the shapes to other users. In the part of chatting, TCP is used for group chatting and private chatting.

When the program starts, the user will be asked to specify the IP address and port number to initialize the TCP protocol, which will be used in the part of chatting. Also, the name of the user will also be requested from the user to make it easier to distinguish. The IP address that the user inputted at the beginning of starting the program will also be used for RMI implementation, which is essential for the drawing board part. The specific interaction diagram is shown below.

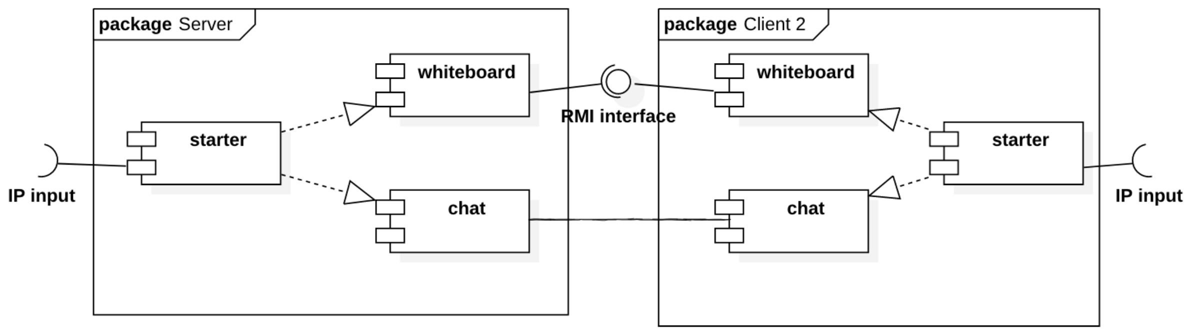

Figure 3: Interaction diagram

3.3 Specific design

3.3.1 The implementation of RMI

In this design, we use RMI to synchronize between different users' whiteboards and keep them update to data. The RMI server is running on the board creator's computers, and others including the creator will create their own RMI client to communicate with this RMI server.

When a user draws a specific graph on the white board, this information will also be transferred to server by a create model method in the RMI client and use the server to broadcast this action to other users. This mechanism has also been implemented in other actions like moving a graphic.

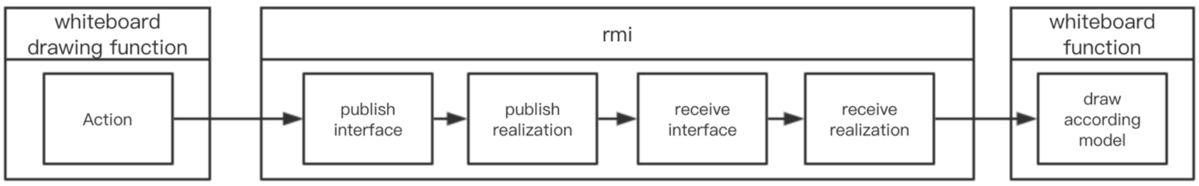

Figure 4: RMI Implementation diagram

3.3.2 Chat window

The chat window is implemented using Java Swing. Specifically, we use JTextArea in Swing as the chat input box, use the getText () method to get the content entered by the user, and use JTextPanel to render the chat content.

This program allows all current users of the system to broadcast messages in the whole group or chat in private. All messages are first sent to the server, which then forwards the message to the client. If the user chooses to chat privately, the server sends it to a specific client; If the user chooses to broadcast, the server forwards the message to all clients.

In addition to the basic text chat function, users can also select emoji or use the Chinese-English translation function in the input box.

3.3.3 Drawing whiteboard

The drawing whiteboard allows multiple users to draw on the whiteboard simultaneously. There are plenty of components in the drawing whiteboard part. We will mention the way of implementing some of them below.

- Drawing Rectangle

When the user presses the drawing rectangle button and then clicks on a specific location on the whiteboard, the system will draw a rectangle on that location. Before drawing, the system will ask the user to choose whether the user wants to draw the filled shape or not. Then, the system will draw the rectangle by calling the functions in the package java.awt.Graphics and the colour of the shape that the user wants to draw can be changed by pressing the colour button on the whiteboard. If the user clicks that button, a window that allows the user to pick from 16 different colours will appear and the user can pick any one of them they like. There are three basic drawing functions applied with RMI, the “add model”, “move model” and “transform model” function.

“Add model” function will be called when a user creates a brand-new model. Obviously, there are somethings we need to pay attention when broadcasting a new model.

Adding a model twice to one user is not allowed. However, RMI will broadcast the model to itself. The solution gives model an ID which is comprised of user ID and model ID. When model is created, a unique id is given to that model and broadcasted with model. If the received model contains the same ID, it will be ignored.

The diagram below displays a concise sequence of how a model is added.

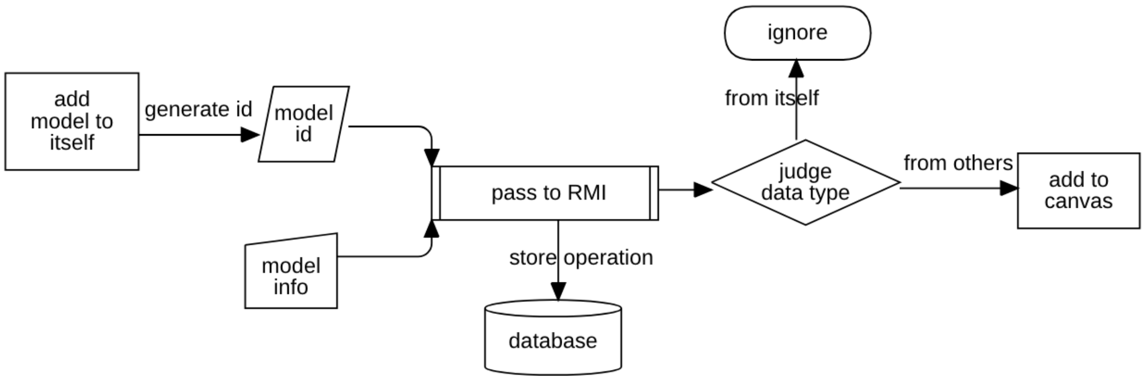

Figure 5: Model Adding diagram

For this project, we assume that the model can only be moved once after its creation by creator. In order to realize it, the simplest way is to control selection rectangular. Thus, according to model ID, if it is null, the selection rectangular will not appear. In terms of moving, we need to judge whether the cursor chooses the model by computing the cursor's location with rectangular position. After dragging a model, new location will be propagated to all the users with model ID. So, every user can refresh the canvas with model in different position.

There is one thing we need to pay attention. "Pencil" and "Eraser" function has really similarity to model moving, cause as cursor goes, the line model need to be added to canvas. So, when these two functions are selected, we need to remove normal mouse drag listener replacing with new listener that can add new models.

"Transform model" also relies on mouse drag listener. The difference is that the transform will happen if only drag the knobs. when this action occurs, the new knob and its diagonal one will be passed to all users. According to these two points, a new size can be drawn.

➢ Open and Save File

The program has a function to open and save the file. This function is only accessible to the manager. When the manager presses the save file button, the system will ask the manager to select a location on the hard disk where he or she wants to store the file. After selecting, the system will save the contents on the whiteboard into the file. The saving process will be based on XML encoder. The models on the whiteboard will be first encoded by XML and then saved into that file. The process of opening file is quite similar to that of saving file. In the opening file procedure, the system will ask manager to select a file he or she wants to open. Then, the system will use the XML decoder to decode the file that the manager has chosen. After that, the system will first clear all the contents that have already exist on the whiteboard and then draw the models that have been decoded from the file. Note that after successfully opening file, the models on the whiteboard of the manager will be broadcasted to other users so that all users are able to see the identical image of the whiteboard.

➢ Save File as Image

There are two different functions in the save as part. One is to save the models in the whiteboard as another file, and the other is to save the figures in the whiteboard as an image. In this project, the format of the image that can be saved is limited to the JPG format. In the process of saving models as an image, we have used the functions in the package javax.imageio to paint the models in an image on the hard disk. Similarly, before saving the models as an image, the manager will be asked to select a location where he or she wants to store the image.

4. Summary

4.1 Critical Analysis

There are quite a few aspects that we have already considered when implementing this distributed shared whiteboard system. First of all, we need to ensure all the users can draw on the whiteboard and chat with each other simultaneously. To make this function reliable, we have designed the multi-threaded server and clients. Specifically, we have used the thread pool to implement the multi-thread attribute of them. Furthermore, it is necessary that all the network transmissions should be reliable. To make it come true, we have used both TCP protocol and RMI technique in this project so that the scalability and the execution performance of network communications can be improved. In addition, we still need to consider that the system should not have any evident delay between server and clients, which is quite essential in this project. We have used the JSON format to transmit data between users to reduce the possible delay between users.

However, we have found some bugs during the process of implementing our system, which is inevitable. To tackle these bugs, we have spent a lot of time learning the cause of these bugs and the solution to them, which is quite helpful to the implementation of the project.

4.2 Conclusions

In this project, we used Java programming language to design and implement the distributed shared whiteboard system, which consists of multi-threaded server and clients. During the process of designing and implementation, we used the multi-threaded method, TCP protocol and RMI technique to make data transmission reliable between server and client. After we finished the project, we have become familiar with multi-threaded programming, socket programming and the implementation of RMI, as well as the method of designing GUI in Java. The performance of the distributed shared whiteboard system is satisfactory, and we are able to put it to use so that we can share the screen as well as chat with each other. In the actual experience, the delay of the whiteboard is very low, and basically it can be synchronized in real-time. Users can use it to draw simple product prototypes and brainstorm and this whiteboard have unlimited possibilities.

4.3 Improvements and Prospects

The distributed shared whiteboard system has been designed and implemented in this project, which has satisfied the requirements of this assignment and can be helpful for people when they share a whiteboard with others. However, this project is far from perfect and there are still a few of ways that the system can be improved.

First of all, in the chatting module, the message transmitted in the network has not been encrypted, which is quite dangerous. Attackers are likely to steal the message that the users sent to others easily. Furthermore, we have only implemented sentence translation between two languages, and it would be better if more languages can be translated between each other in this project.

5. Innovations

5.1 A wide variety of fonts

Our whiteboard provides users with a wide variety of fonts. This feature allows users to insert text using different fonts, which is very personalized.

5.2 Erasers of different sizes

Our whiteboard provides users with a total of five different sizes of erasers: very small, small, medium, big and very big. This feature allows users to erase more efficiently, which is very practical.

5.3 Translation

Users can use the Chinese-English translation function in the chat window, which facilitates users in different languages to communicate and better use the whiteboard together.

5.4 Emoji

Users can select and send emoticons in the chat window, which enriches the functionality of chat. Emoji is a common language that spans languages, and emoticons that incorporate body language and emotions are more touching than simple words.

5.5 File Integrity Verification

In this project, I have used the SHA-256 function to verify the file so that we can ensure the integrity of the file when the manager chooses to store the file. This is to prevent someone else from modifying the file that is stored in the hard disk. When the manager wants to save the drawing models on the hard disk, the system will first generate a file to save it and then calculate the hash value of the file, which will be stored in another file. When the manager chooses to open the file that has been previously saved in the hard disk, the system will verify the integrity of the file by comparing the hash value of the file with the hash value that is stored in another file. If these two value matches, the system will successfully open the file. Otherwise, the system will notify the manager that the file has been destroyed. To make the verification process reliable, we have chosen SHA-256 as the hash function to verify the files, as it is more secure than MD-5 and SHA-1.

5.6 Night mode

We have implemented dark mode in this project to allow each user to choose a dark version or light version according to their preference, since it is a very popular and user-friendly function, and can protect the users’ eyesight.