90015 Distributed Systems

Assignment 2: Distributed Shared White Board

Haichao Song
854035
May 2020
University of Melbourne

Table of Contents

Problem Context

In this project, a shared whiteboard which allows multiple users to draw simultaneously on a canvas is designed and implemented using client-server architecture. A single central server manages all client's states and the communication between each other. The server uses Java RMI framework. Remote interfaces and servants are designed to make the communication between server and clients. Threads are the lowest level of abstraction for network communication and concurrency.

Component Description

Remote

Remote contains the IServer Interface which is shared between the client and the server. It includes all the functions provided by the system to the client.

Server:

The server side is composed by the RMI server controller, the board handler, the remote server and the server GUI.
The RMI server controller is responsible for 1) creating an instance of the remote server and publishing the instance in the rmiregistery 2) creating the server GUI.
The board handler is responsible for 1) defining all the attributes needed in the board information communication 2) handling the events happening on the board.
The remote server is the server-side implementation of the remote interface. It extends Unicast Remote Object to allow the JVM to create a remote proxy/stub.
The server GUI shows the address and port number to users.

Client:

The client side is composed by the client controller, the client GUI, the event handler, the user list, board panel, user panel and a package of modes clients can choose.
The client controller is responsible for connecting the IServer and create the client GUI.
The client GUI allows users to draw on the whiteboard and contains board and user panel to allow users perform actions on them.
The event handler is responsible for 1) continuing calling to get board events to update board events from server side 2) handling the events on the white board and showing the drawing to the local client 3) displaying and updating user list
The user list is responsible for defining the basic color setting of the user list.
The board panel extends JPanel and is responsible for 1) defining the menu on the white board 2) handling functions on the menu 3) call add board events to send the event to the server.
The user panel extends JPanel and is responsible for 1) defining the user list on the white board 2) handling function for manager to kick users 3) refreshing and updating the user list 4) call add board events to send the event to the server.
The "Modes" package includes all modes clients can choose to draw on the white board. Each class is responsible for defining the methods of drawing that mode on the white board. It also contains an enum class which defines all the modes and their representative characters.

Design Diagram

Server

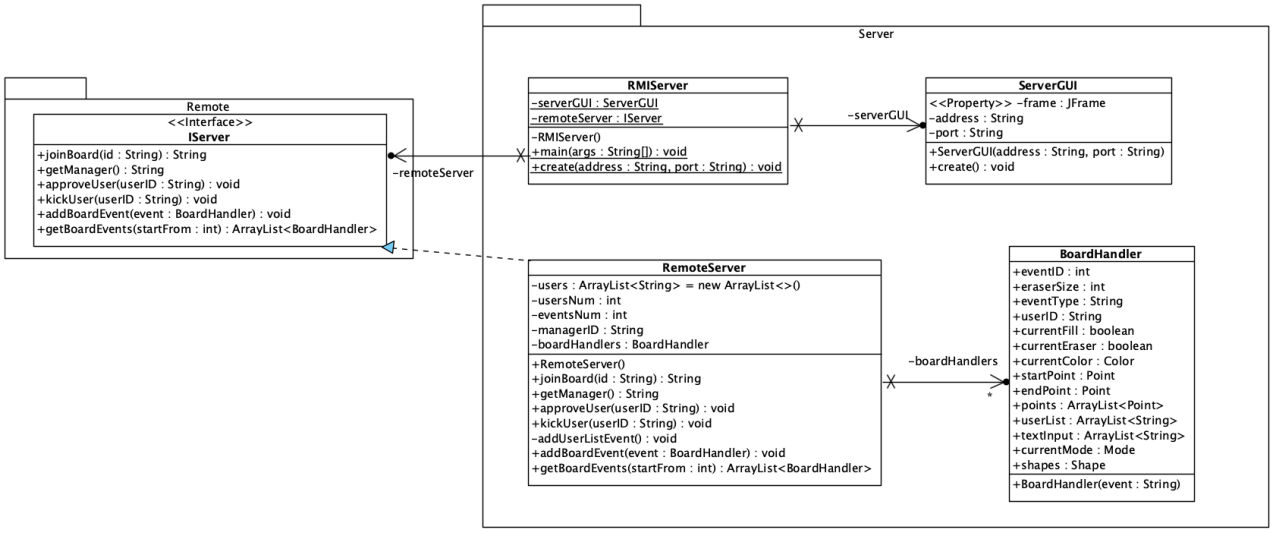

Figure 1. Server-Side Class Diagram

The class diagram (Figure 1) shows the components of the server with the attributes and methods inside. Once the server starts, it checks the input argument format and then creates an instance of the remote server as well as the server GUI (Figure 2 and Figure 3.). The remote server implements IServer and creates the board handler to handle events sent from the client side, which contains information about the changes happening on the white board. The Server GUI will be shown in the next page (Figure 4).

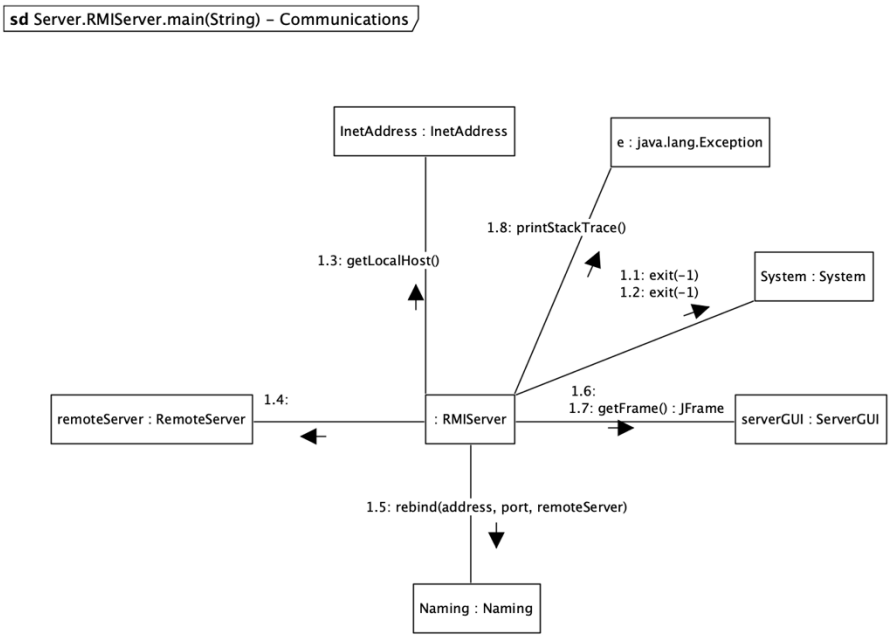

Figure 2. RMIServer Communication Diagram

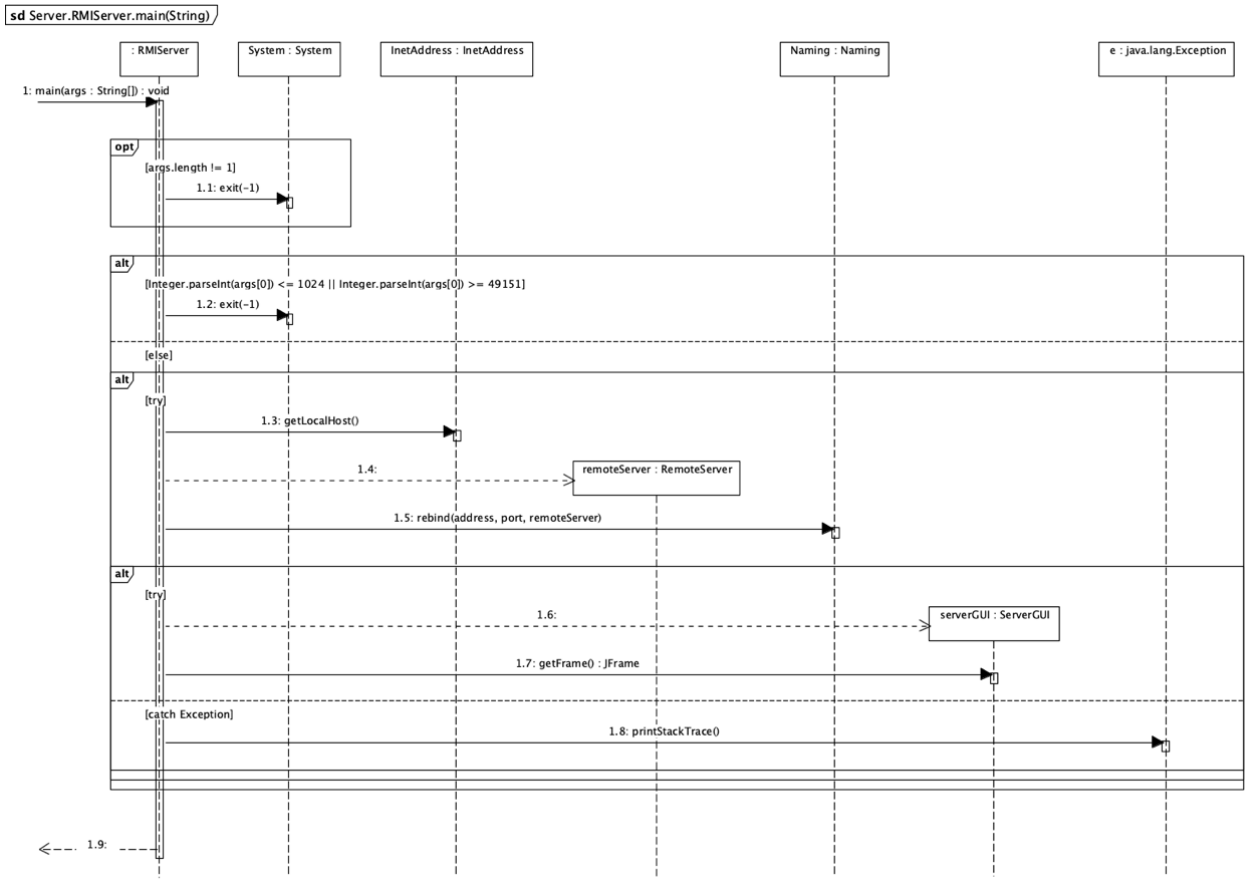

Figure 3. RMIServer Sequence Diagram

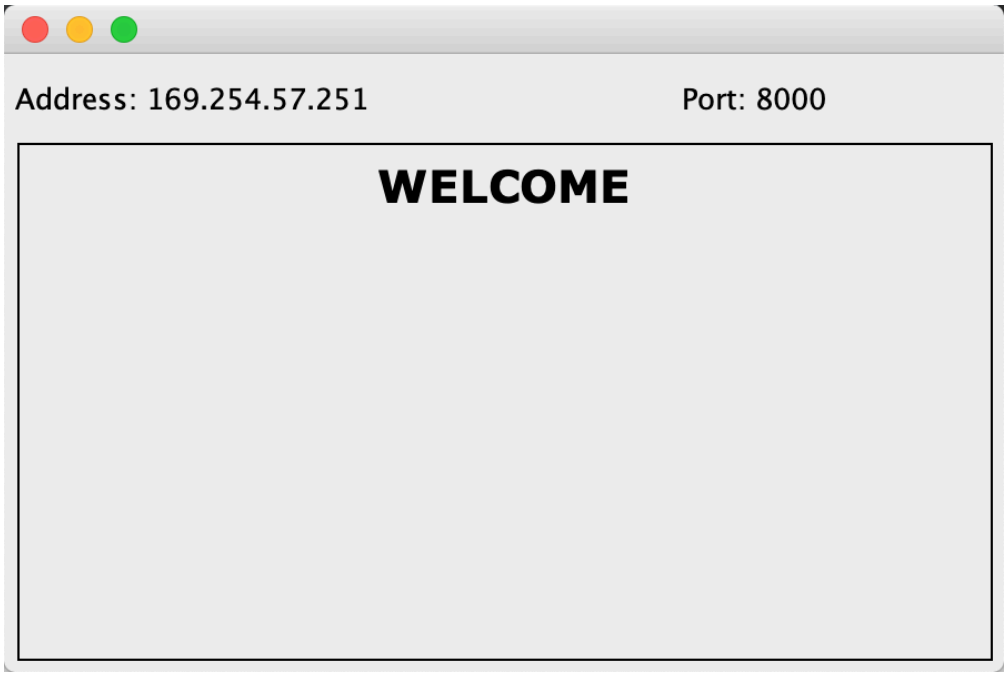

Figure 4. Server GUI

Client

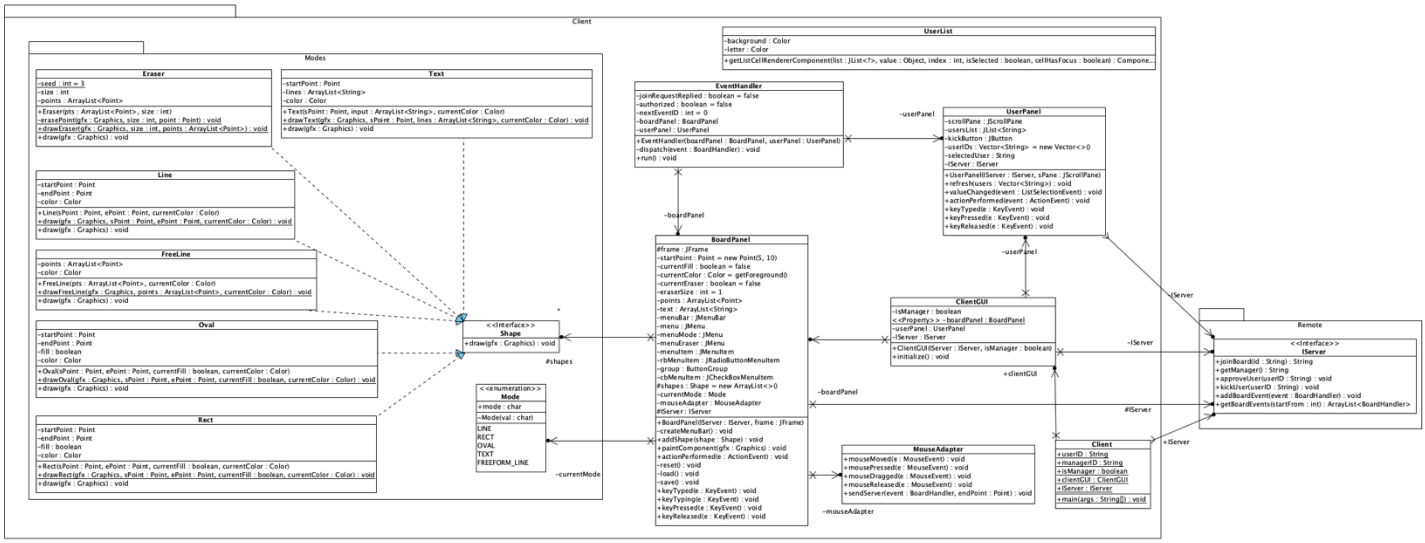

Figure 5. Whole Client-Side Class Diagram

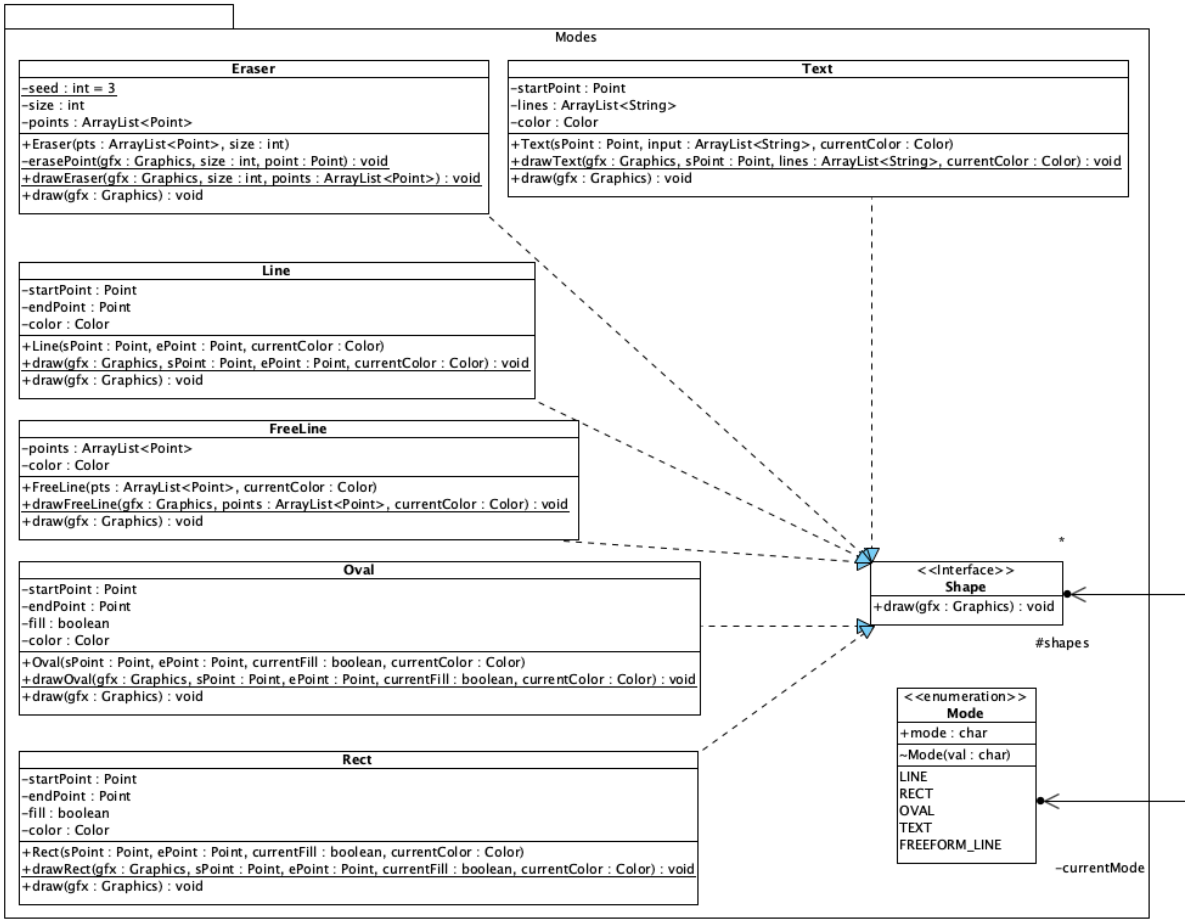

Figure 6. Mode Package Class Diagram

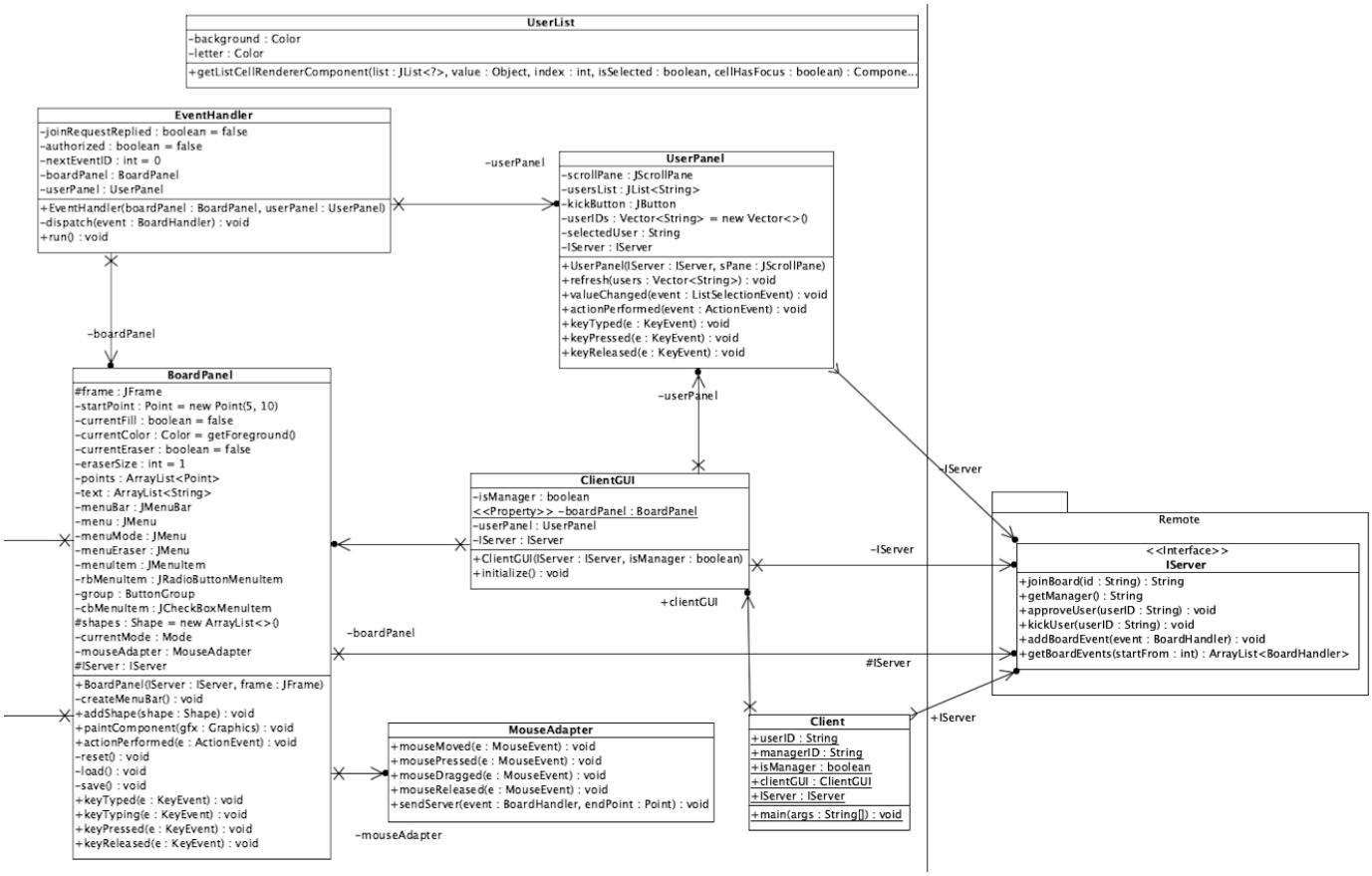

Figure 7. Other Part Server Class Diagram with Remote

The client package contains a number of classes (Figure 5). They are mainly divided into a package of modes (Figure 6) and other classes including event handlers, user and board panels, client controller and the client GUI (Figure 7).

Once the client starts, it checks port number by look up naming and gives users a window to fill their usernames (Figure 8). After that, it will create a client GUI and allow operations by users. The board panel is listening to all the operations on the board and the menu (Figure 9), and the user panel is listening to all the operations on the user list (Figure 10). They will distinguish different events and call add board events to send the event to the server. After the server updates, the event handler keeps calling to get board events from the server and dispatch the event on the local white board (Figure 11 and Figure 12).

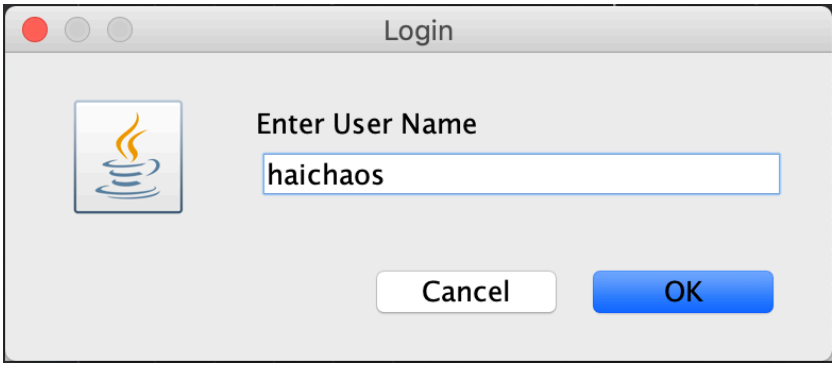

Figure 8. GUI for Entering Username

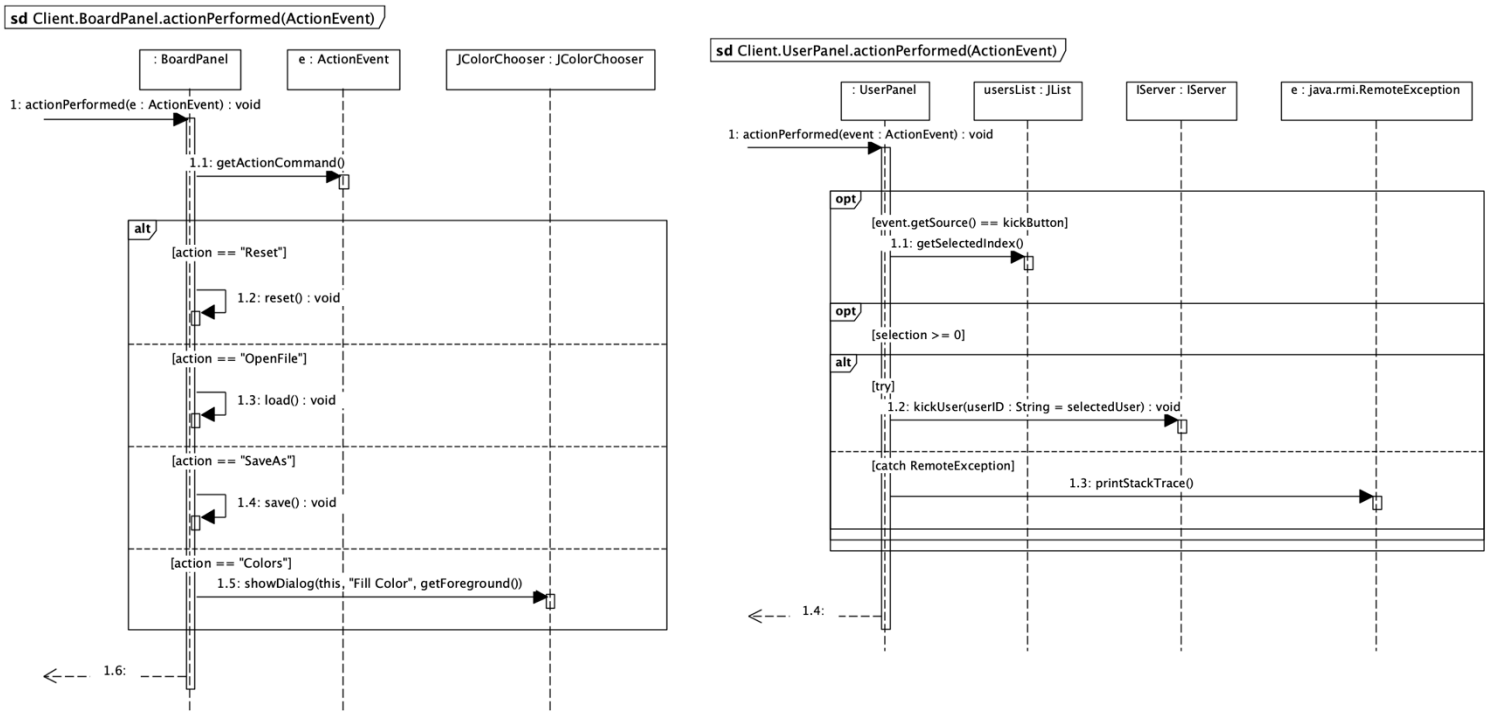

Figure 9 & 10. BoardPanel & UserPanel ActionPerformed Sequence Diagram

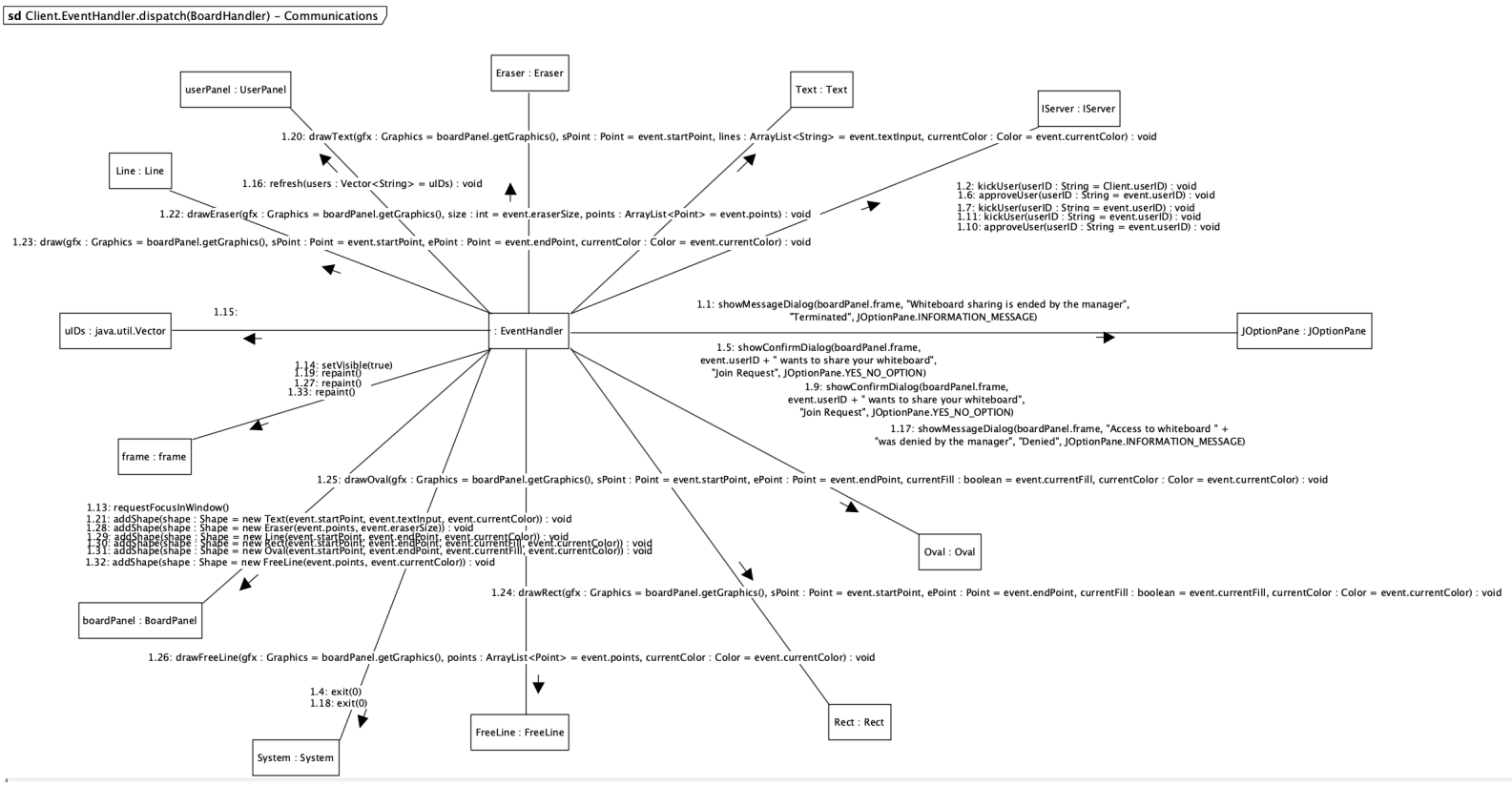

Figure 11. EventHandler Dispatch Communication Diagram

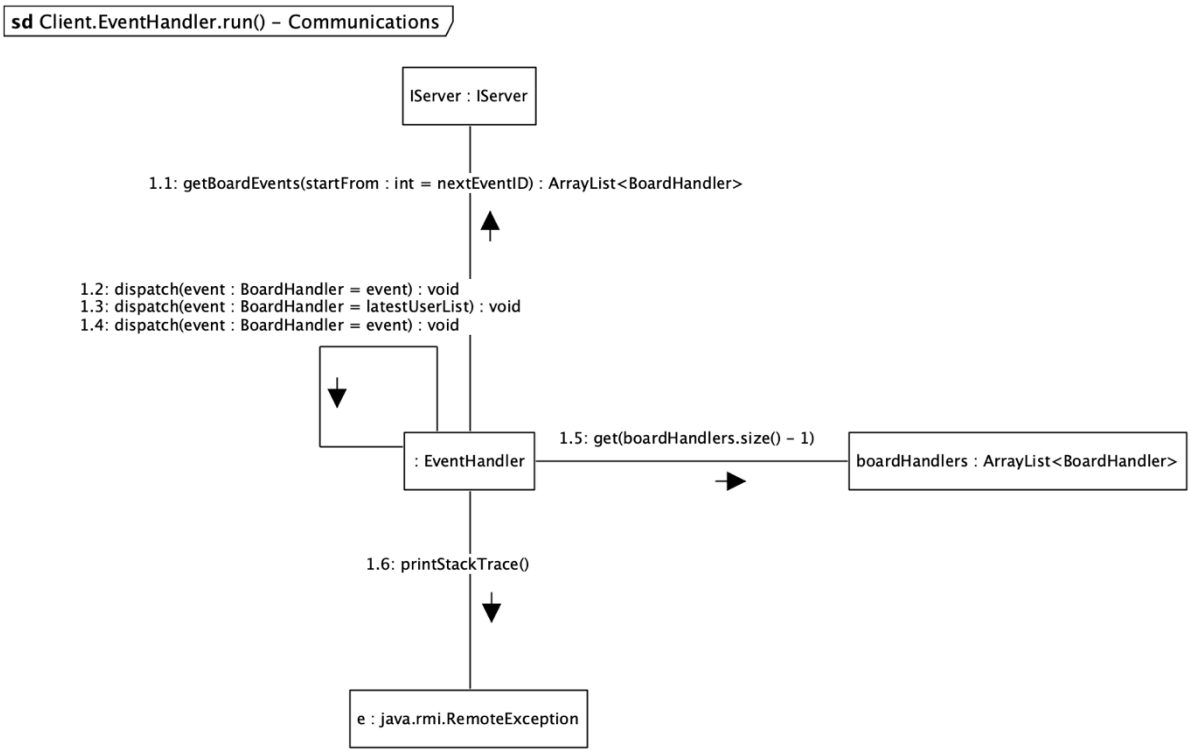

Figure 12. EventHandler Run Communication Diagram

The sequence diagrams of 5 main functions in both board panel and user panel will be shown below (Figure 13 - Figure 17).

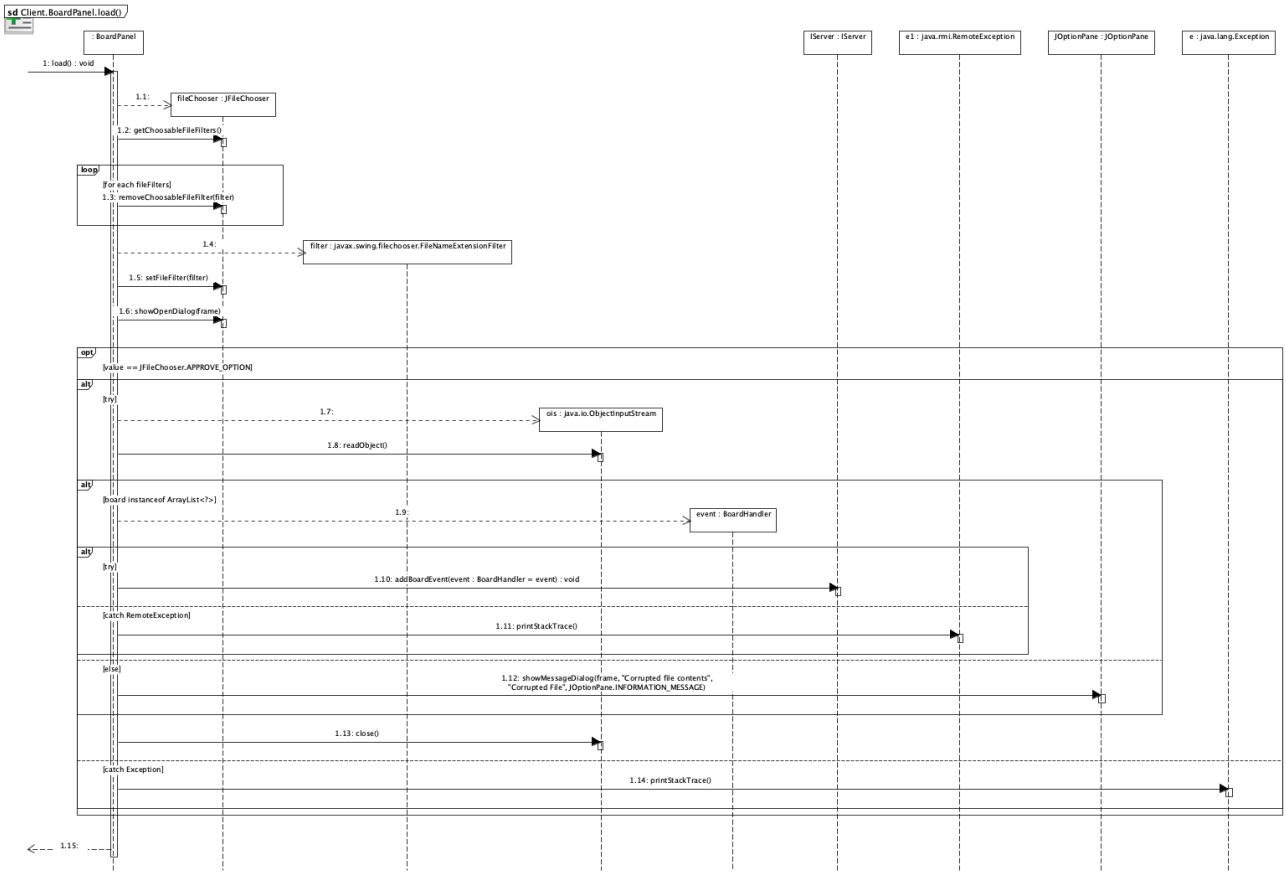

Figure 13. BoardPanel Load Sequence Diagram

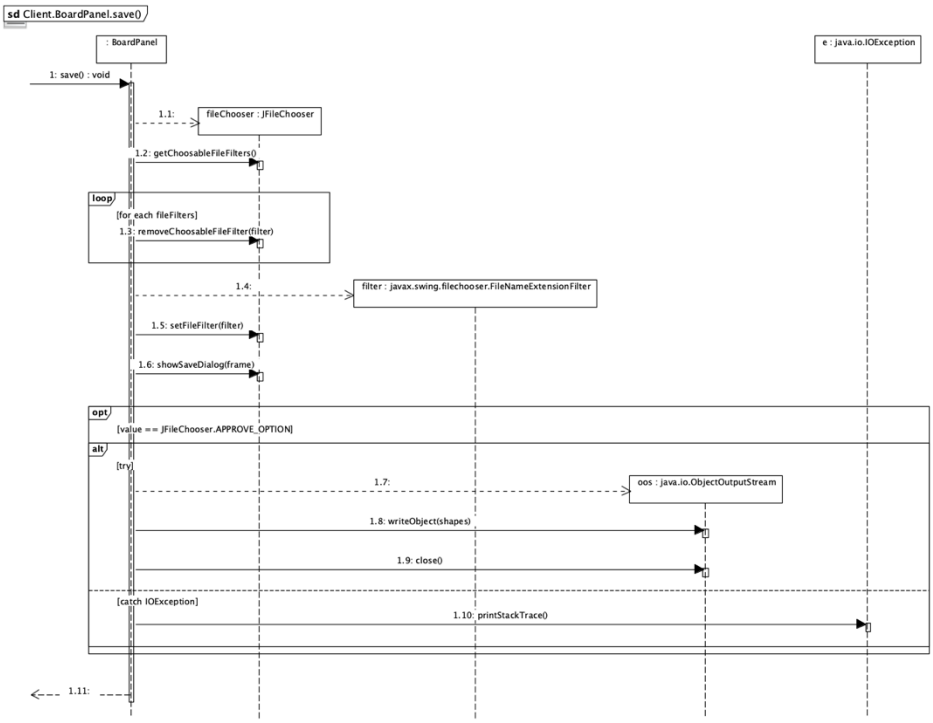

Figure 14. Board Panel Save Sequence Diagram

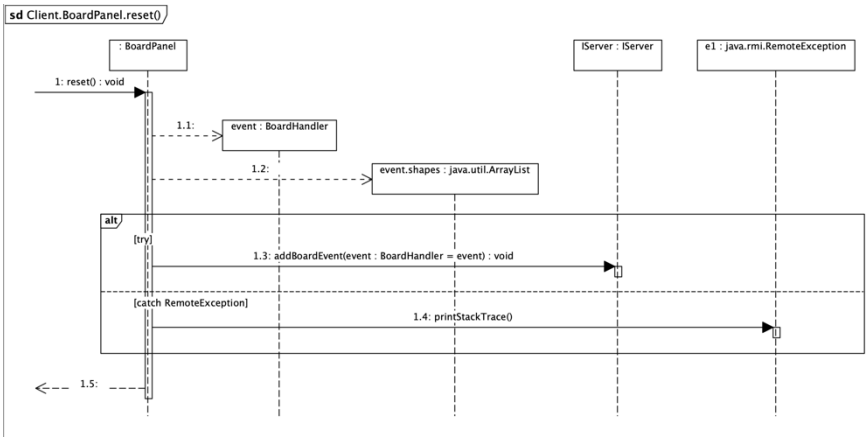

Figure 15. BoardPanel Reset Sequence Diagram

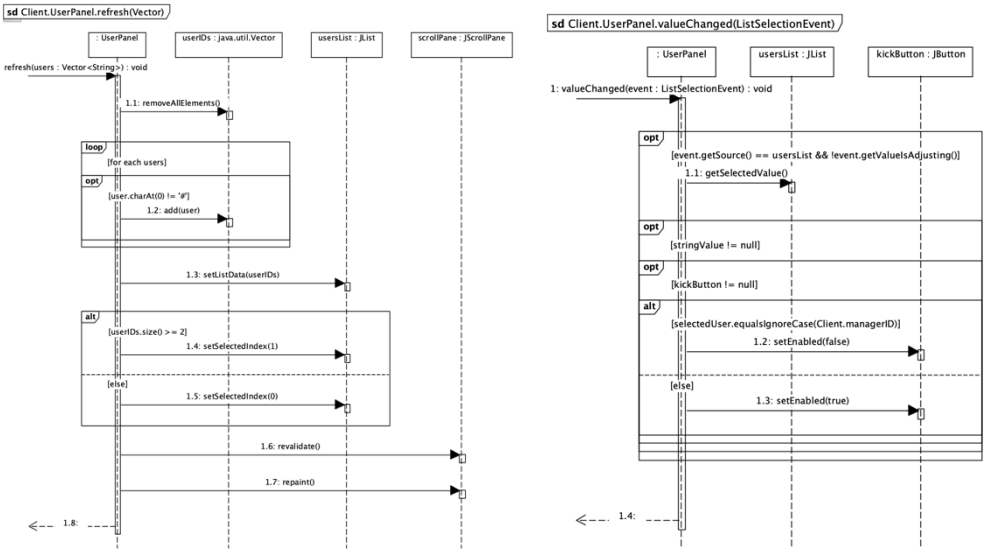

Figure 16 & 17 User Panel Refresh & ValueChanged Sequence Diagram

The Client GUI has a menu bar on the top of the white board. It allows the manager to save, load, reset or close the current white board. The “Modes'' button contains different modes users can draw on the white board. Users can also choose colors and fill in the “Option” by clicking the option button. Finally, the eraser can erase the drawing in certain places.
The User list is on the left side of the white board. It shows all clients sharing the current white board. The manager who enters the white board first has the right to kick other users out.

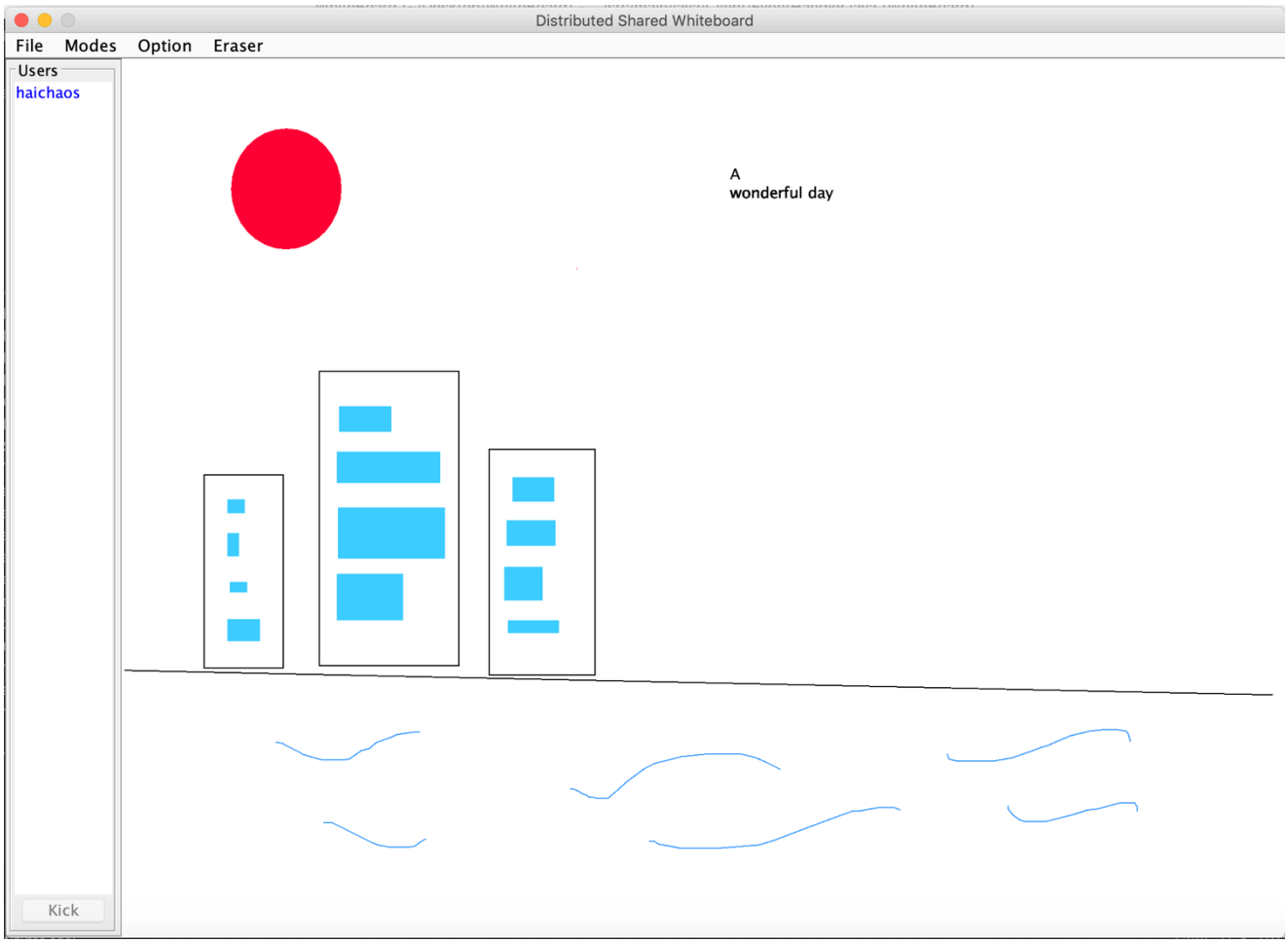

Figure 18. Client GUI White Board

Critical Analysis

System Architecture

The project implements client-server architecture. A single central server manages all clients and the board status. All clients will get the board events from the server and render the events on their clients' white boards.

The client-server architecture has following advantages:

1. Integration of Service: Every Client access the corporate information shared by the server. This feature is very fit for our project since clients can corporate drawing on white board and the same drawing is sharing among all clients.
2. Improved Data Sharing: Data is retained by usual business processes and manipulated on a server is available for designated users (clients) over an authorized access [1]. It supports open access from all clients and is transparency in network service.
3. Handle Different Platforms: Applications used for clients/servers are built regardless of hardware platform. Different clients on different platforms can connect to our server and share the white board.
4. Easy Maintenance: client-server architecture is a distributed model representing dispersed responsibilities among independent computers integrated across a network. It is easy to replace, repair, upgrade and relocate a server while clients remain unaffected.
5. Security. A single central server has full control of all clients and resources can ensure the data manipulation authority and be administrated efficiently.

It also has some disadvantage:

1. Server Overload: Since a single server needs to handle all clients' requests, it might result in traffic congestion. In order to solve this problem, we can use a proxy server on the client side to reduce server loads and improve performance.
2. High Impact of Error. Since there is only one server in the architecture, if the server fails the whole system will break down. It can be avoided by building multiple servers and running servers on different host computers.

Compared to peer-to-peer architecture, although client-server architecture is more expensive for implementing the central server, it has more security and is more stable when traffic increases. According to the above analysis, client-server architecture is more fitful for our white board project so that I choose to implement this system architecture.

Communication Protocol

The project uses Java RMI framework as the communication protocol. Compare to the Socket APIs which are the low-level abstraction that Java applications interact with network. RMI is a framework and protocol family for implementing application-level networking between Java applications. [2]

The RMI has the following advantages:

1. Handle threads and Sockets for underlying communication. RMI handles its own network connection so that there is no need to consider low-level communication problems.

2. Easy to write. Remote Server is easy to implement. It only needs an interface and a several lines remote server. This helps for quick server construction and quick function testing.
3. Server can be modified and recompiled without considering the interface and the client side. RMI server and client are individually connected to the remote interface and implement interface to connect to each other. Therefore, two sides are relatively independent. Changing the code in a method will not make any influence on the other side.
4. Good network Security. RMI uses built-in Java security mechanisms that allow your system to be safe when users download implementations [3]. It protects systems and networks from hostile codes.
5. Heterogeneity. The RMI based system is portable to any JVM. The code user RMI can be run on any Java Virtual Machine.

It also has some disadvantage:

1. Strictly limited to Java. Since it is a Java API, it depends on JVM. It cannot be used with other languages.
2. Security needs to be monitored. Although RMI has good network security, the dynamic class loading can still be overridden by the third party and cause secure problems.
3. Overhead of marshaling and object serialization [4]. The RMI provides marshalling objects and serialization for users, but it also decreases the efficiency of RMI compared to Socket APIs.

According to the above analysis, Java RMI still provides a lot of benefits for our project so that I choose to use RMI as the communication protocol.

Error Handling

Most errors which may happen in the process of using are properly handled. They could be mainly divided into the following parts:

1. Input parameters error:

Error	Description
Insufficient parameters	Input arguments less than the number required
Invalid parameter	Input arguments are not correct type
Invalid port number	The port number is not between 1024 - 49151

2. Operation Error:

Error	Description
File not find	Open an inexistent file
I/O error	Error format when reading file, request, response

3. Connection Error:

Error	Description
Remote exception	failures of remote method calls
Class not find exception	JVM wants to load an inexistent class
Unknow host exception	IP address cannot be determined
Malformed URL exception	no legal protocol could be found
Not bound exception	lookup or unbind an unassociated binding

Innovation

1. Server GUI. The server GUI is provided to show the address and port number to the user.
2. File and Manager Operation. Advanced features are implemented. Manager can open, save and reset the whiteboard as well as kick out other users.
3. Color and Fill. Users can choose the color to draw different modes on the whiteboard and choose to draw filled rectangular and oval.
4. Free Line. Users can choose “Free Line” mode and draw freely on the whiteboard.
5. Eraser. Users can use eraser by choosing eraser on the menu and cleaning the drawings on certain places of the whiteboard.

Conclusion

Overall, by implementing suitable architecture and communication protocols as well as handling a series of errors. Our system is:

1. Heterogeneity. As mentioned in both system architecture and communication protocol, both of them can be supported on different hardware platforms.
2. Failure Handling and Transparency. Most errors in the process of using are properly handled in the system.
3. Scalability. Client-server architecture is stable and scalable for providing services to a number of clients.
4. Concurrency. Users share the same whiteboard. All events happening on the whiteboard can be shown instantaneously to all users.
5. Reliability and Openness. Both system architecture and communication protocols provide good security standards.

Beyond the performance of the system now, there is still space for further improvement. Multiply server and proxy can be considered; more functions can be added to the whiteboard and security standards can also be improved.

Reference

[1] MediaWiki, “Client Server Architecture”, CioIndex, *https://cio-wiki.org/wiki/Client_Server_Architecture*, Advantages and Disadvantages of the Client Server Architecture, Aprill 20 2020.
[2] Stephen C, “rmi vs servlets vs sockets”, Stark Overflow, *https://stackoverflow.com/questions/5798473/rmi-vs-servlets-vs-sockets*, Aprill 27 2011.
[3] Oracle Technology Network, “Java Remote Method Invocation - Distributed Computing for Java”, *https://www.oracle.com/technetwork/java/javase/tech/index-jsp-138781.html*.
[4] Web NMS, “#110072 - Advantages and Disadvantages of RMI calls over TCP”, *https://kbase.zohocorp.com/kbase/Web_NMS/Communication/file_110922.html*.