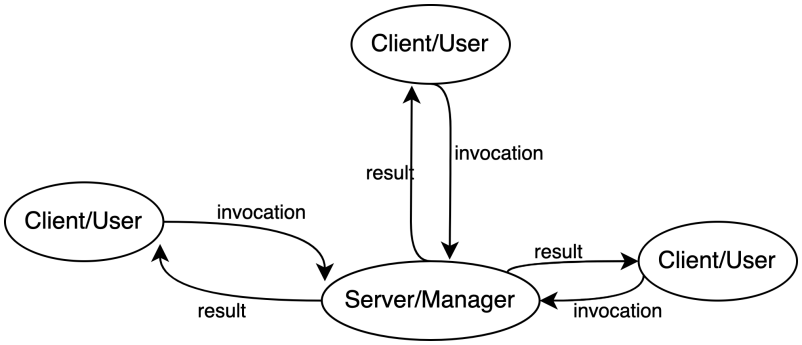
COMP90015 Assignment 2 Report

Student Name: Qingyang Feng Student ID: 980940

System Architecture



The system follows a classic client-server architecture. The system has one single central server that manages all the system state. The first user starts a server and becomes the manager of the created whiteboard. Other users can make requests to connect to the server and join the whiteboard. Users can draw line, circle, oval and rectangle as well as print text on the whiteboard with various different colors and they can also message in the ChatBox. Other than the users' functionalities, the manager can accept/reject users' request to join the whiteboard; kick out a user; create a new whiteboard and open, save, saveAs a file. All data is stored in the server including the shapes drawn on the whiteboard, the message history as well as the list of usernames.

Communication Protocols and Message Formats

The remote communication in the distributed system is handled by two technologies: sockets and RMI.

The communication for dealing with clients' joining requests is handled by TCP sockets. When the server is launched, it binds a server socket to the provided port and listens for incoming socket connection requests. If an incoming socket connection request is accepted, the server will create a new socket to serve the client's request to join the white board. During the whole time the server is up, the original socket will be open and keep listening for new connection requests. Messages are sent as strings in the socket. The client will first send their username to the server through the socket. The server will check whether the username has already been

used or not. If the username has already been used, the server will send a message "`userNameAlreadyInUse`" to the client and on the client's side a dialog window will popup to ask for a new username. If the username is unique, the server will notify the manager for approval of the request to join the whiteboard and then send a "`approved`"/ "`rejected`" to the client. If the client received "`approved`", a shared whiteboard will be created for them. Otherwise, the client will receive a notification saying they are rejected to join the whiteboard. After the client's join request is completed, the socket will be closed for the client.

After the client has joined the whiteboard, all other communication is achieved by Java Remote Method Invocation (Java RMI). The communication protocol used by Java RMI is Java Remote Method Protocol (JRMP). Serializable objects are used as parameter or return data and object serialization is used to serialize the data for transmission between the server and clients. The shapes drawn on the canvas, the message history and the list of users are stored in the remote object. Both the server and clients are able to get the state of the whiteboard and make updates to it with different privileges by invoking the methods of the remote object.

Class Design

The overall class design is as follows:

- **CreateWhiteBoard**: the main class that starts the server, creates the manager whiteboard GUI and creates new threads to handle users' join requests with a thread-per-request model.
- **JoinWhiteBoard**: the main class that makes request to join the server whiteboard and launches a user whiteboard GUI if the request is approved
- **JoinRequestThread**: the class that extends the Java `Thread` class to handle users' join requests
- **(Package) WhiteBoard**: includes the classes that implements different component of the WhiteBoard GUI
    - **WhiteBoard**: the manager/server WhiteBoard GUI
    - **Canvas**: the whole canvas that includes the toolbar, the drawing area, the userlist and the chatbox.
    - **MenuBar**: the menu bar for operations available for the manager of the whiteboard and is only present in a manager whiteboard GUI. It has a file menu

including the ‘new’, ‘open’, ‘save’, ‘saveAs’ operations and a user management menu including the ‘kick a user out’ operation

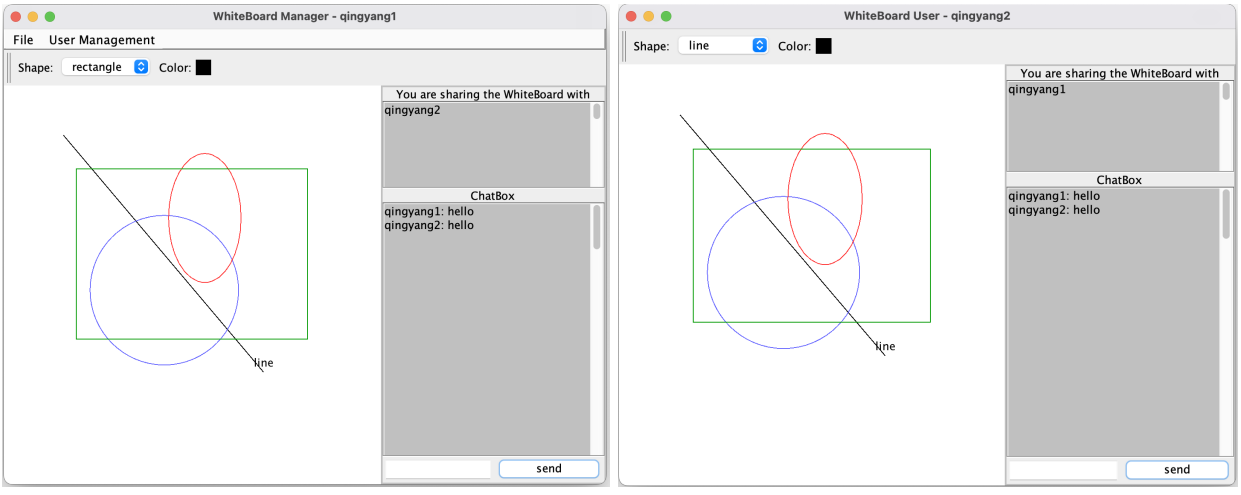
    - **ToolBar**: the tool bar that allows users/manager to choose the shape or color they want to use
    - **SidePanel**: the panel to show on the right side of the GUI that displays a list of users that are currently sharing the whiteboard and a ChatBox to send messages and display the message history.
    - **Shape**: a shape to be drawn on the canvas, that is defined by its color, position, type(line, circle, oval, rectangle, text) and content (only available for type text).
- **(Package) RMI:** includes classes that creates and implements the remote interface
    - **IRemoteWhiteBoard**: create a WhiteBoard interface
    - **RemoteWhiteBoard**: implements the WhiteBoard interface with attributes and methods

The class diagram is attached in the Appendix.

Interaction Diagrams

The sequence diagram for requesting to join the WhiteBoard is attached in the Appendix.

Implementation Details



Drawing on the Canvas

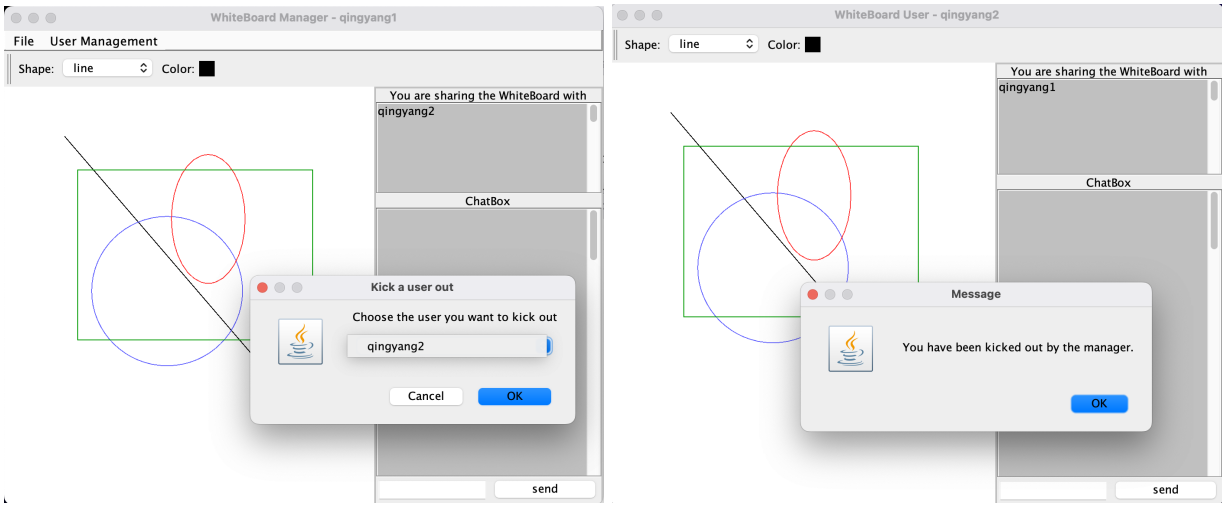
Drawing on the canvas is enabled by implementing MouseListener and MouseMotionListener. To draw a shape on the whiteboard, a user can first choose their desired shape and color from

the toolbar and then click, drag and release on the canvas to create the shape on the whiteboard. To text on the canvas, the user can double click on the position where they want the text to appear and then, in the popup dialog window, type in the text. The WhiteBoard GUI listens to the mouse events from the users, records the shape type, color, startPoint, endPoint, and content (only for text) of the shape drawn and add the shape to the arrayList of shapes stored in the remote object by calling the remote method `addShape(shape)`.

Messaging in the ChatBox

To message in the ChatBox, a user can type the message in the textField and then click the send button. The WhiteBoard GUI listens to the click on the send button, captures the message in the textField and adds the message to the arrayList of messages stored in the remote object by calling the remote method `addMessage(message)`.

Kick a user out



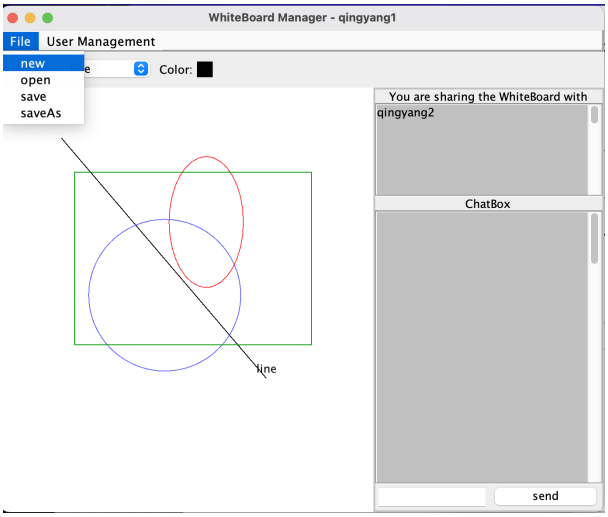
To kick a user out, the manager removes the user from the userlist stored in the remote object and adds the user to the list of kicked out users by calling remote methods `removeUser(username)` and `addToKickOut(username)`.

State update of the Shared WhiteBoard

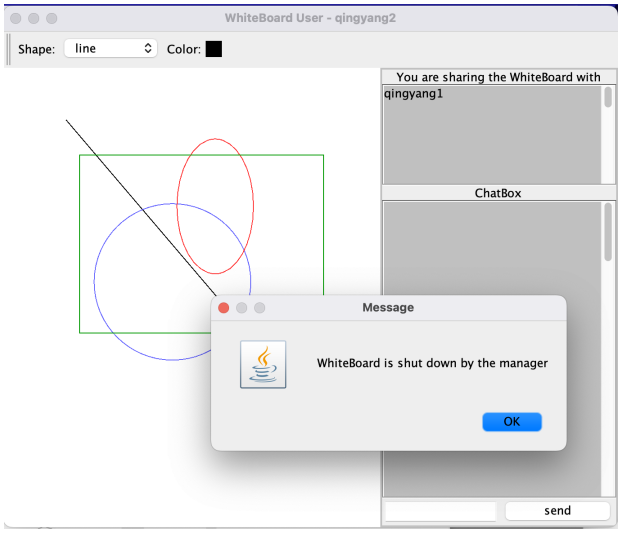
The state of the WhiteBoard GUI for every client/server is updated every 0.1 second. Which means that, every 0.1 second, the current values of the shapes drawn by all users, the chat history, the userlist and the list of kicked out users are retrieved from the remote object by calling remote methods and are updated to the corresponding components of the WhiteBoard

GUIs. This ensures that editions made by any user to the whiteboard are visible to all users without appreciable delays.

File Menu



User Notification when Manger Close the Server



Multithreading

For handling client's join requests, a new thread is created for each successful socket connection while the original thread keeps listening on the original port. The thread is destroyed after the join request is completed. For handling concurrent invocations of the remote methods, Java RMI uses the default thread-per-request model. Java RMI creates a new thread on the

server for each remote invocation that executes the method call and returns the result to the client.

Analysis

All the basic features and advanced features are implemented.

Advantages and Disadvantages of Java RMI Compared to Socket:

Advantages:

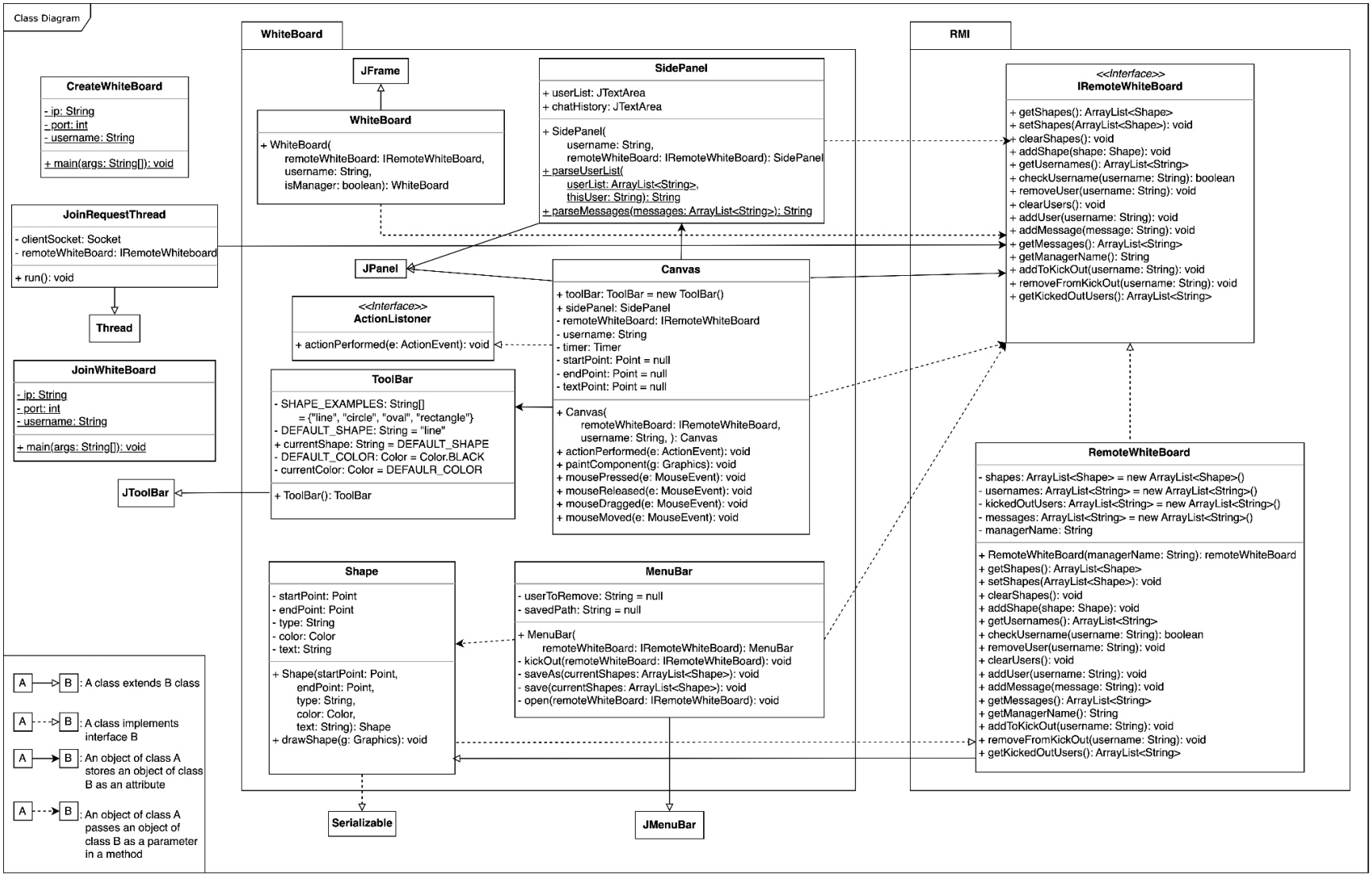
- Automatic handling of multithreading
- Automatic handling of the underlying communication protocol
- Automatic marshaling and unmarshaling making it much more easier to pass complex objects between different machines in the distributed system compared to using socket programming
- Easier implementation

Disadvantages:

- More complex configuration compared to socket programming
- Additional computational overhead due to Automatic marshaling and unmarshaling which may affect the performance of the application when large data is transmitted

Appendix

Class Diagram:



The sequence diagram for requesting to join the WhiteBoard: