COMP90015 Distributed Systems
Assignment 2 Distributed Shared White Board

Jiaming Xian 1336110

1. Project Background

The shared whiteboard allows multiple users to draw simultaneously on a canvas. This system implements the functions of drawing *lines*, *circles*, *ovals*, *rectangles*, inserting *text*, and has options such as selecting *colors*, *strokes*, and *fonts*. This system also implements user management functions including *view* and *kick*, and a chat room located on the right side of the interface. In addition, this system also provides file operations for managers, including *new*, *open*, *save*, *save as*, and *close*.

2. System design and components

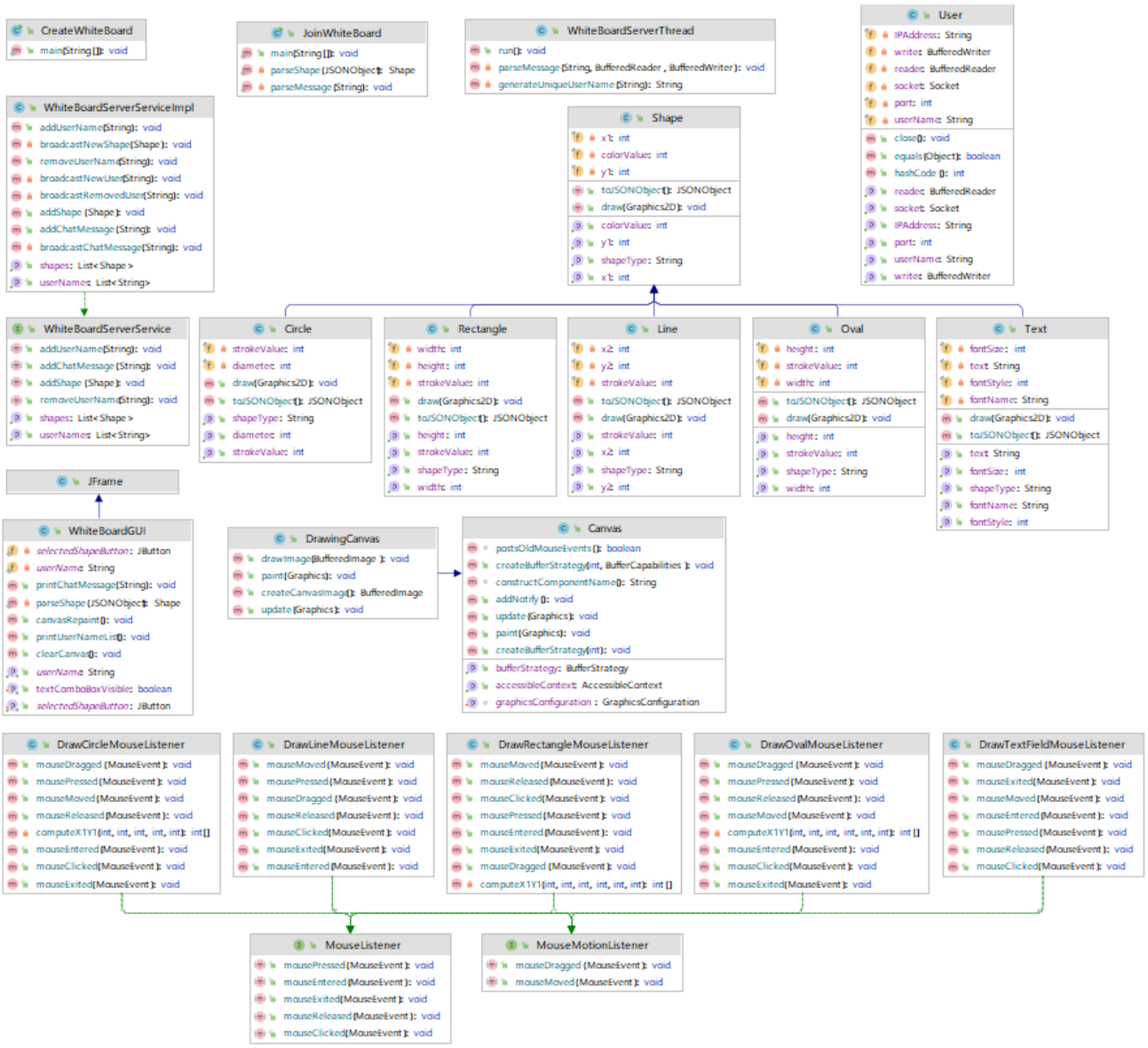

Figure 1. White Board System Class Diagram

CreateWhiteBoard: used by manager to create a white board server process.
JoinWhiteBoard: used by clients to join a white board process.
WhiteBoardThread: extends Thread class, used by server to create threads for each connection.
User: a User class object is created for each client.
Shape: an abstract class to be extended by classes of specific shapes.
Circle, Line, Rectangle, Oval, Text: Classes of specific shapes that extends Shape class.
WhiteBoardServerService, WhiteBoardServerServiceImpl: remote methods and corresponding implementations used in Remove Method Invocation (RMI).
WhiteBoardGUI: used to create Graphical User Interface (GUI) for server and clients.
DrawingCanvas: extends Canvas class to create a drawing canvas on GUI.
Drawing(Shape)MouseListener: attached to shape buttons for drawing operations.

3. Multi-thread and Concurrency

This system implements multi-threading and allows multiple users to access it simultaneously. Each incoming client establishes a socket TCP connection with the server. The server loops to listen for new connection establishment requests and establishes a service thread for each new connection. A thread is created by instantiating a WhiteBoardServerThread object.

This system also ensures concurrency. In RMI, each method of adding/removing graphics or users or chat information is declared synchronized, ensuring block synchronization when the method is being called simultaneously. The function for updating the whiteboard in the WhiteBoardGUI class is also declared synchronized. Also, the method of parsing Json information in the socket communication thread also ensures concurrent synchronization.

4. GUI

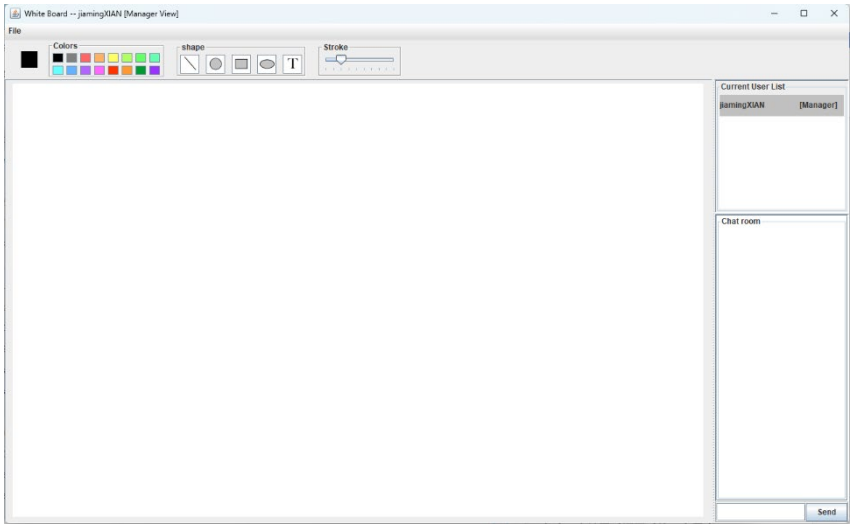

Figure 2. White Board GUI

In the center of the GUI is canvas, above which are the color panel, graphics panel,

stroke panel, font panel, and then the file menu. On the top right of the GUI is a list of current users displayed, where the manager can view and kick users. At the bottom right is a chat room where you can see messages from other users and send your own messages.

For the manager, you can operate the new, save, save as, and close items in the file menu. The whiteboard file will be saved in. json and. png formats, and can be opened in. json format.

5. RMI and Socket

This system uses a combination of RMI and Socket. Each participant maintains a shapeList to record the current shapes on the canvas, and a userNameList to record the names of the current participating users. The manager additionally maintains a userList that contains the User objects established for each customer, which contains the user's input/output streams and personal information.

5.1 RMI

For RMI, the following methods are provided.

`List<String> getUserNames()`: Each user needs to obtain a list of current user names when joining.
`List<Shape> getShapes()`: Each user needs to obtain a list of current shapes when joining. And every time the shapes list is refreshed (open a file), the server will notify the user to call this method.
`void addUserName(String userName)`: Each user needs to add his username to the current list and call the broadcast function broadcastNewUser() when joining.
`void removeUserName(String userName)`: When each user exits (actively or kicked), they need to remove the username from the list and call the broadcast function broadcastRemovedUser().
`void addShape(Shape shape)`: When each user draws a new shape, add the new shape to the shape list and call the broadcast function broadcastNewShape().
`void addChatMessage(String chatMessage)`: When each user sends a new message in the chat room, the broadcast function broadcastChatMessage () is called.

The private broadcast functions are as follows:

`private void broadcastNewUser(String userName)`: Traverse through the userList, obtain their write stream, and inform them of the new username they need to add.
`private void broadcastRemovedUser(String userName)`: Traverse the userList, obtain their write stream, and inform them of the username that needs to be removed.
`private void broadcastNewShape(Shape shape)`: Traverse the userList, obtain their write streams, and inform them of the new shape.
`private void broadcastChatMessage(String chatMessage)`: Traverse the userList, obtain their write streams, and inform them of the new chat room message.

5.2 Socket

For sockets, each client maintains a continuous TCP connection with the server. When the client requests to join the whiteboard, first establish a socket connection and wait for the server's consent. The server establishes a service thread for each socket connection to achieve multithreading. Afterwards, the socket connection will continue until one party exits.

During this period, certain operations will trigger socket communication, mainly through the invocation of broadcast functions, which are based on sending messages to the write stream of each user socket in each connection.

And other operations can also trigger socket communication, especially when the client's operation needs to notify the server, because broadcasts cannot be broadcasted to the server (userList only records the clients). At this point, it is necessary to inform the server of the operation through the socket. For example, if a user wants to add a new shape, he can invocate the remote method to broadcast, but at the same time, they must also inform the server to update the drawing board in a timely manner through the socket.

In addition, the file operations for manager only do not trigger RMI, so it is necessary to use the socket communication channel to inform users to update the shapeList.

6. Information Exchange Protocol

The information exchange protocol on sockets is JSON. Each set of information on the socket communication channel is a JSON object. Both the server and client have their own parsing functions to parse JSON command messages and perform corresponding operations. To achieve JSON format exchange for the Shape class, the subclasses of Shape implements a toJSONObject() method. Clients can restore Shape objects by using a provided parsing method. Otherwise, Shape objects will not be serialized and transmitted.

The reading and writing of whiteboard files are mainly in JSON format. Saving the whiteboard file will generate a .json file, and a .png image file with the same name for visual browsing. To read a whiteboard file, it must be a. json file. The toJSONObject() method is implemented in the concrete subclasses of the Shape class, allowing for the conversion of Shape objects to JsonObject. This makes it possible to convert the shapeList to a JsonArray for saving as a. json file. When reading, parse JsonArray into a shapeList.

7. Creativity Points

7.1 Preview Graphics

This system implements preview graphics function. When drawing graphics, preview lines can help users predict the size and position of the graphics to be drawn, thereby better controlling the drawing process.

When dragging the mouse to draw, you can preview the shape that is about to draw. When you release the mouse, it will be drawn on the canvas.

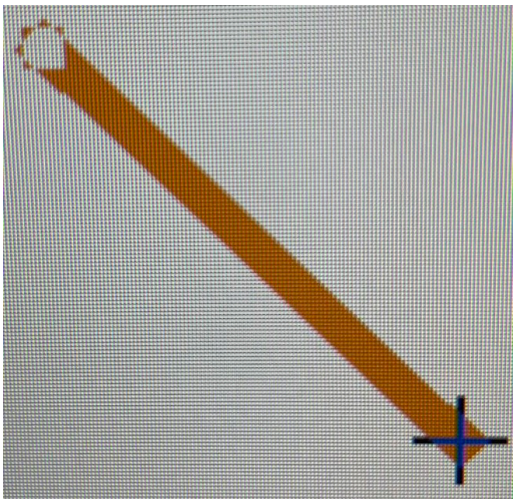

Figure 3. Preview Graphics Function Demonstration

7.2 Stroke and Font Tools

This whiteboard provides a stroke slider and drop-down selection boxes for font, style, and size related to text.

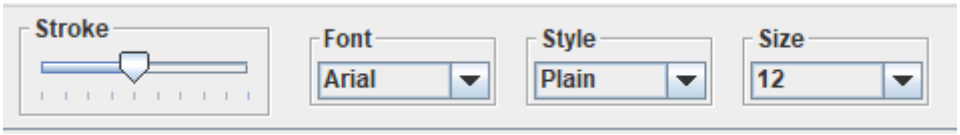

Figure 4. Stroke and Font Tools

7.3 Advanced Features

This system implements all the required advanced features, including *new*, *save*, *save as*, *close* file operations, a chat room, and a kick user operation.

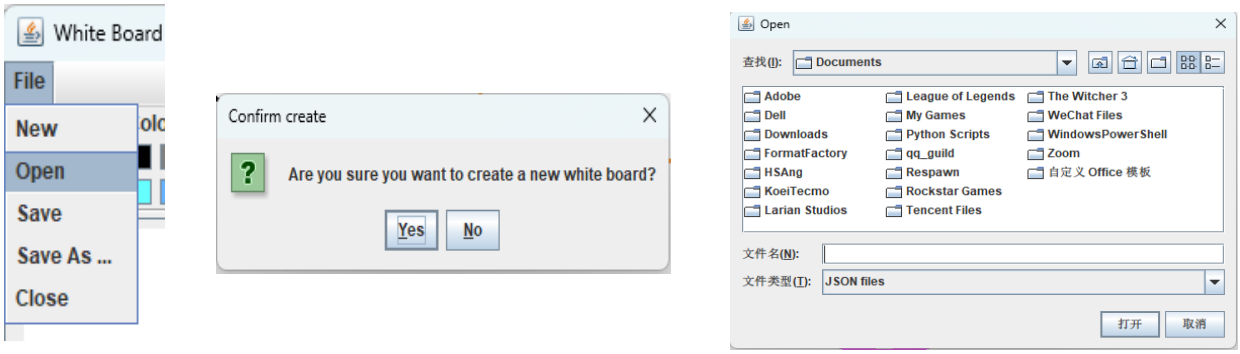

Figure 5. File Operations

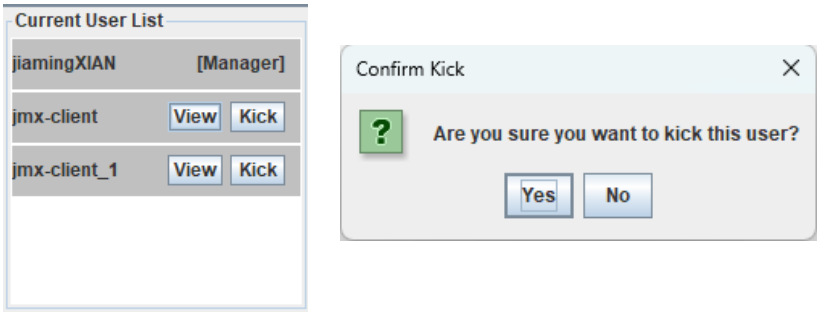

Figure 6. View And Kick Functions

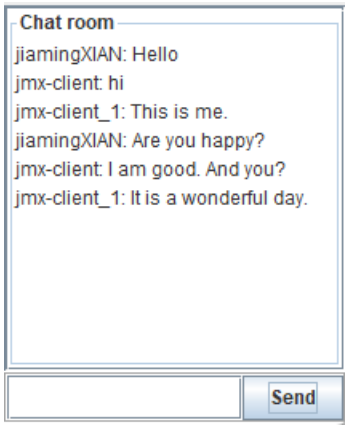

Figure 7. Chat Room Snapshot

7.4 Security Channel – Encryption and Decryption

This system encrypts the transmitted messages and decrypts the received messages, achieving secure communication in distributed systems. Security is based on AES symmetric encryption, where the server and client hold the same secret key.

```
Encrypted Message:mouYcw4zcJ+JEyZozbzB3dmo1Tq/bgyiFDO1zUzeFNE2dNwdsj9gIBFyqwwr5wHX
Decrypted Message:{"response":"acceptJoin","userName":"jmx-client"}
Decrypted Message:{"Shape":{"strokeValue":3,"y1":119,"x1":169,"y2":232,"colorValue":-1
Decrypted Message:{"Shape":{"diameter":171,"strokeValue":6,"y1":222,"x1":230,"colorVal
Decrypted Message:{"response":"kick"}
Encrypted Message:0X4T1CYBKg3ZhR3XrYIsniu+I0btQXS7stlt3ALnJfg=
```

Figure 8. Secure Message Transmission

8. Conclusion

The design and implementation of this system can meet the basic features and advanced features of the multi-users whiteboard application based on a Client-Server architecture. The system effectively implements synchronous concurrency and also designs two user-friendly GUIs.

In the future, the system can consider adopting a worker-pool architecture, that is the system instantiates a certain number of threads to make them available, but this is based on statistics of user scale.

Also, the system can be further optimized for data format, such as using database technology to store the shapes/canvas as a database. Server-side operations are

performed directly on the database. Database technology can better ensure the format constraints, integrity, and consistency constraints of data, and provide low-level data management encapsulation to provide more abstract services to the software layer. Software design could be able to pay more attention to top-level design and service provision.