Distributed System and Application

Zhihao Liang
1367102

1. System Architecture

The whiteboard system is a Java application. Each module represents a different important part of the application. CreateWhiteBoard.java and JoinWhiteBoard.java are the starting points for the whiteboard manager and other users, respectively. Meanwhile, WhiteboardServer.java is responsible for running the server-side operations.

The controller package contains classes responsible for the control logic of the system, such as WhiteBoardController.java. WhiteboardCanvas.java handles the drawing canvas of the whiteboard.

In the model package, various classes represent different entities and constants in the system. MessageType.java and OperationType.java define different types of messages and operations, while Role.java represents different user roles.

The network package has two sub-packages: client and server. The client package contains classes responsible for handling different aspects of client-side communication, such as ClientConnectionHandler.java and ClientServer.java. The server package handles server-side communication, containing classes such as ServerConnectionHandler.java and BroadcastExecutor.java.

The operations package contains classes representing various drawing operations that users can perform on the whiteboard, such as drawing a line, circle, oval, rectangle, and text.

Finally, the utils package includes utility classes and functions, such as FileManager.java for file operations, DialogUtil.java for dialog operations, and MessageBuilder.java for building messages.

2. Communication Protocols and Message Formats

The shared whiteboard system relies on Netty, a non-blocking I/O client-server framework in Java, for establishing connections and managing communications between clients and servers. Netty uses the power of low-level network programming while providing a high-level API, thereby efficiently handling complex networking operations and maintaining high performance.

The communication between clients and servers in the system occurs through JSON-formatted messages, making the system highly interoperable and data easily human-readable. Google's Gson library is used to convert Java Objects into their JSON representation and vice versa.

Each JSON message contains a type field indicating the type of the operation. For instance, a type of 'join' or 'leave' denotes a user's action of joining or leaving the whiteboard. The userId field represents the unique ID of the user performing the action. When the type is 'join,' an additional role field is included to indicate whether the user is a 'manager' or a 'member.'

The type field also defines operations performed on the whiteboard. For drawing operations such as 'DRAW_RECTANGLE,' 'DRAW_CIRCLE,' 'DRAW_OVAL,' 'DRAW_LINE,' and 'DRAW_TEXT,' there are additional fields describing the specifics of the operation such as coordinates, dimensions, color and text content.

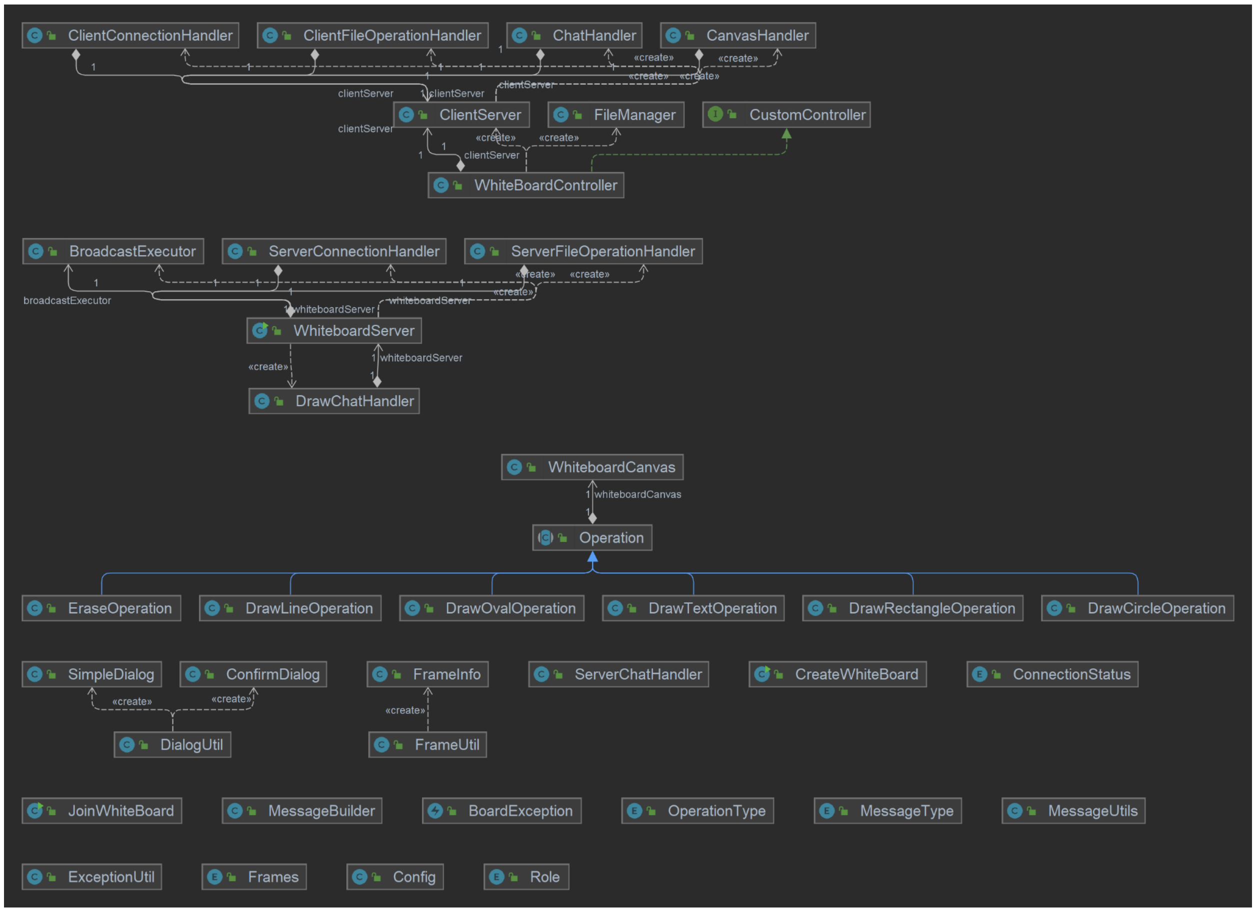
Additional message types such as 'kick,' 'approval,' and 'chat' cover management actions, approval statuses, and chat messages respectively. For instance, a 'kick' message also includes the targetUserId and the reason for the kick.

The 'userList' message type returns a list of active users. The 'snapshot' type deals with opening files and has a field commands for the commands to execute on the file, while 'writeToBoard' denotes whether the content should be written to the board.

Lastly, 'fileOperation' types cover actions such as 'new,' 'save,' 'close,' and 'open' on a file. For the 'open' operation, it carries an array of commands similar to the 'snapshot' type.

3. Design Diagrams

a. Class Diagrams



4. Implementation Details

a. GUI Implementation

The GUI for the whiteboard has been created using JavaFX, which is a Java library commonly used to build rich internet applications. It comes with a graphics pipeline for rendering primitives and has a variety of pre-built UI controls, including buttons, text fields, and more complex components such as tables, trees, menus, charts, and so on.

The UI layout has been defined using FXML, which is a scriptable, XML-based markup language used for creating JavaFX application user interfaces.

In the Java controller class (WhiteboardController.java), different mouse events are handled for drawing on the canvas:

onMousePressed: When the mouse is pressed on the canvas, this method is called. If the selected shape is text, it creates a new Text shape at the clicked position. If it's not text, then it starts a new path at the clicked position.

onMouseDragged: When the mouse is dragged over the canvas, this method gets called. It creates a new shape and adds it to the list of shapes based on the selected shape type. If the shape type is a line and the mouse movement passes a particular threshold, it sends a 'draw line' message to the server.

onMouseReleased: This method is triggered when the mouse is released on the canvas. It adds the final shape to the list of shapes and sends a draw message to the server depending on the selected shape type.

JavaFX's ColorPicker is used for color selection. The selected color is used as the stroke color when drawing shapes on the canvas.

For text input, there's a TextField where users can enter the text. This text is used when the selected shape type is text.

The ListView in the right VBox displays a list of active users and chat messages. Users can type and send messages using the TextArea and 'Send' button.

b. Networking

i. Serve-side:

ServerConnectionHandler is responsible for managing user connections and the overall state of the users in the system. When a user attempts to join the server, the ServerConnectionHandler first checks for any duplicate users. If a user with the same ID is already present, it sends a disapproval message. Otherwise, the user is added to the corresponding group (either 'managers' or 'waitingMembers') based on their role.

If the message is a manager's approval, the ServerConnectionHandler allows the member to join, updates the member list, broadcasts the updated list, and sends a welcome message. It also synchronizes the canvas data for the new member. If a member is not approved, the ServerConnectionHandler informs the member and removes them from the waiting list.

When ServerConnectionHandler receives a 'kick' message from a manager, the ServerConnectionHandler removes the kicked user from the members' list and closes their connection. For a 'leave' message, it removes the user who is leaving.

DrawChatHandler appears to handle messages related to drawing and chat operations. For drawing operations, it adds the operation to the 'historyDrawCommands' list and broadcasts the operation. For chat messages, it simply broadcasts the received message to the manager and peers.

ServerFileOperationHandler handles operations related to the whiteboard's file system. This includes creating a new board, saving the current board, closing the board, and opening a saved board. It uses MessageUtils to identify the type of the message. For 'newBoard', 'saveBoard', and 'closeBoard' messages, it broadcasts the message after the operation. For 'openFile' messages, it clears the 'historyDrawCommands', populates it with the commands from the opened file, and then broadcasts the message.

ii. Client (peer and manager) side:

ClientConnectionHandler is responsible for handling incoming messages related to the user's connection status. It handles 'kick out' messages by setting the client status as 'kicked out' and closing the connection. If it's a manager approval message, it checks whether the user has been approved or not, and updates the client status accordingly. if not approved, it displays an alert dialog with the given reason for rejection.

ChatHandler is focused on dealing with chat-related messages. When a chat message is received, it sets the chat message in the client server. If it receives a user list message, it parses the user list from the message and updates the client server's user list.

CanvasHandler processes incoming messages related to canvas operations. If the operation message is from the client themselves, it adds the message to the 'savedCommands'. Otherwise, it executes the drawing command received. It also handles 'snapshot' messages by parsing the drawing commands from the message. If the snapshot is meant to be written to the board, it executes the commands on the client's canvas.

FileOperationHandler handles messages related to file operations on the whiteboard. It handles 'newBoard' messages by resetting the canvas and clearing the 'savedCommands'. 'closeBoard' messages result in the client's connection being closed. 'openFile' messages are handled by parsing the drawing commands from the message, resetting the canvas, clearing and updating the 'savedCommands', and executing the commands on the canvas.

5. Conclusion

I have implemented all the features, including basic and advanced features. The project successfully enabled multiple users to interact simultaneously on a shared canvas, incorporating functionalities such as drawing shapes, inputting text, selecting colors, and providing user-to-user communication through a chat window.

Managing concurrency was a major obstacle. However, overcoming this challenge improved my comprehension of managing shared resources and controlling concurrency.

This project gave me an extensive experience in developing applications, enhancing my understanding of designing interfaces, managing network communication, implementing multi-threading, and managing user interactions. Overall, this project has been a rewarding experience, improving my current skills and broadening my knowledge in the field of distributed system design.