Assignment 2-Distributed System and Application

Name: Ninghao Zhu(Bruce)

Student ID: 1446180

School of Computing and Information Systems, The University of Melbourne

COMP90015: Distributed System

May 18, 2025

Problem Context

The task for Assignment 2 is to design and develop an online whiteboard system that allows multiple users to draw on the same whiteboard simultaneously and chat with each other via text. In addition, it also requires the design and development of user management and file management features, enabling manager to manage users and the file.

System Architecture

WeDraw adopts a client-server architecture, allowing multiple users to connect to a multi-threaded server simultaneously for drawing and chatting. On the client side, the main components include WhiteBoardGUI, ClientGUI, and WeDrawClient. Users connect to the server through WeDrawClient, use ClientGUI for identity verification and login, and interact with WhiteBoardGUI, which consists of the whiteboard and chat panel, drawing tools panel, user panel, and file dropdown menu. On the server side, the main components include Command, Data, Request, Threads, UserManagement, as well as IClientCallback, IWhiteBoard, RemoteClientCallback, RemoteWhiteBoardServant, and WeDrawServer. The first user to connect is identified by the server as the manager, while subsequent users are recognized as players. The manager has the authority to manage users and files. Both administrators and players can draw on the whiteboard and participate in chat. The server receives drawing and chat commands, and then broadcasts them to all connected users.

Communication Protocols

To simplify the process and code required to connect to the server via sockets, WeDraw adopts Java RMI, which allows clients to invoke methods on server-side objects as if they were local. Java RMI is built on TCP, a connection-oriented communication protocol that provides a reliable data stream, ensuring the accurate transmission of drawing and chat information between clients and the server.

Message Formats

WeDraw transmits objects and data using Java's serialization mechanism. In the server's Command folder, the DrawCommand class is defined and implements the Serializable interface. This class encapsulates the drawing mode along with the necessary properties and methods for drawing. Several specific command classes, including CircleCommand, EraseCommand, FreeDrawCommand, LineCommand, OvalCommand, RectangleCommand, and TriangleCommand, inherit from DrawCommand and implement their respective drawing methods. Similarly, the TextCommand class also implements the Serializable interface, and DrawTextCommand extends it to support text-based drawing. These message protocols standardize the data exchanged between the client and server, making the system easier to maintain, extend, and manage.

Design Diagrams

WeDraw mainly consists of two parts: the Client and the Server.

On the client side, the main components include:

WhiteBoardGUI

The WhiteBoard acts as the main container, responsible for loading the other panels and registering mouse events to send drawing commands to the server. The ToolPanel provides various drawing tools for user interaction, while the UserPanel displays the list of currently connected users. The ChatPanel is used to display messages exchanged between users in real time. The FileDropDown is a menu restricted to administrators, with event listeners registered for actions such as opening, saving, and closing files. Together, these components form an integrated and interactive environment for collaborative drawing and communication within the WeDraw system.

ClientGUI

Launch the login interface and verify whether the user is a manager. Then, load the WhiteBoard and synchronize the current whiteboard state.

WeDrawClient

Retrieve command-line arguments and launch ClientGUI.

On the server side, the main components include:

Command

Encapsulates all serialized drawing and chat commands.

Data

Includes DrawMode, WhiteBoardFile, and WhiteBoardState. DrawMode enumerates the all drawing modes. WhiteBoardFile is responsible for loading and saving whiteboard files. WhiteBoardState stores the current drawing state.

Request

Serialization of login result.

Threads

Threads Worker Pool, use Worker Factory to Create Worker, use Worker to execute Request.

UserManagement

Includes UserType, the User class, Manager and Player classes, as well as the UserList.

IClientCallback

This interface defines methods that enable broadcasting messages or commands to all connected users.

Full Name: Ninghao Zhu(Bruce) Student ID: 1446180 WeDraw

IWhiteBoard

This interface includes methods for receiving drawing and chat commands on the whiteboard, synchronizing the whiteboard state, managing the user list, and performing administrative operations.

RemoteClientCallback

This class implements the methods of the IClientCallback interface.

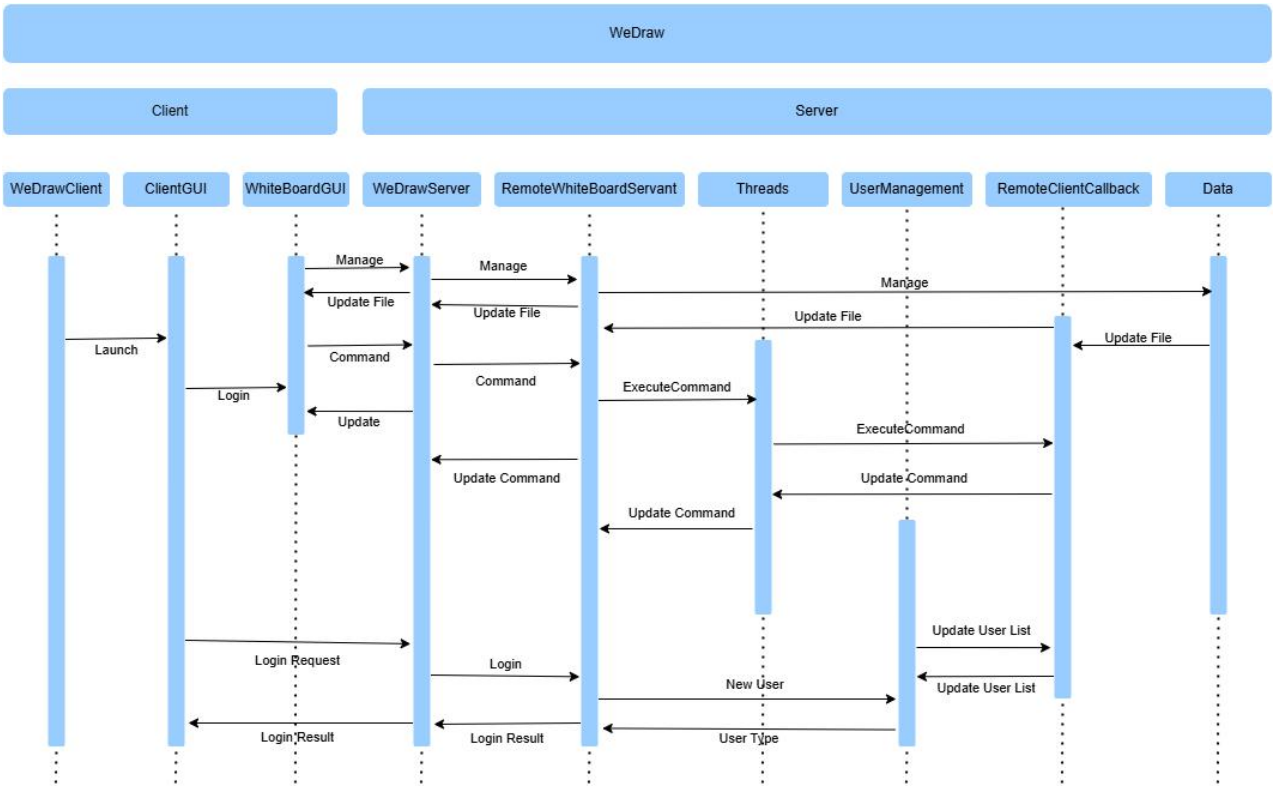
RemoteWhiteBoardServant

This class implements the methods of the IWhiteBoard interface.

WeDrawServer

Start the server service and accept client connections.

Interaction Diagram



Implementation Details

WeDraw uses Java RMI to allow users to invoke remotely implemented objects and methods as if they were local. The following are some key implementation details.

WhiteBoard

Mouse events such as MOUSE_PRESSED, MOUSE_DRAGGED, and MOUSE_RELEASED are registered on the WhiteBoard. When users select different drawing tools, the corresponding events are triggered and drawing commands are then sent to the server.

Canvas Update

When the server receives a drawing command from a client, the WorkerThreadsPool starts a thread to execute it. Inside the RemoteWhiteBoardServant, an instance of WhiteBoardState stores these commands using an ArrayList data structure. Then, the receiveDrawCommand method in RemoteClientCallback is invoked, which calls the drawShape method to synchronize the received drawing command with all users.

Chat Panel

The chat button registers the KEY_PRESSED event. When the user presses a key, a chat command is sent to the server. The subsequent handling process is the same as described above.

User Management Skeleton

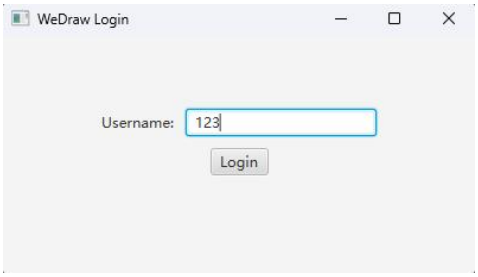
After a user logs in, the Login method in RemoteWhiteBoardServant is invoked. The first user to log in is designated as the Manager, while all subsequent users are identified as Players. The newly logged-in user is added to the instantiated userList. When a Player attempts to join, the Manager is prompted via the ManagerPermit method to decide whether to allow the user to access the whiteboard. The Manager also has the authority to remove users from the userList through a kick-out operation; upon clicking the button, the selected user is notified and removed from the whiteboard. If the Manager exits the whiteboard, all connected users will receive a pop-up notification, and the whiteboard will then close for everyone.

File Menu

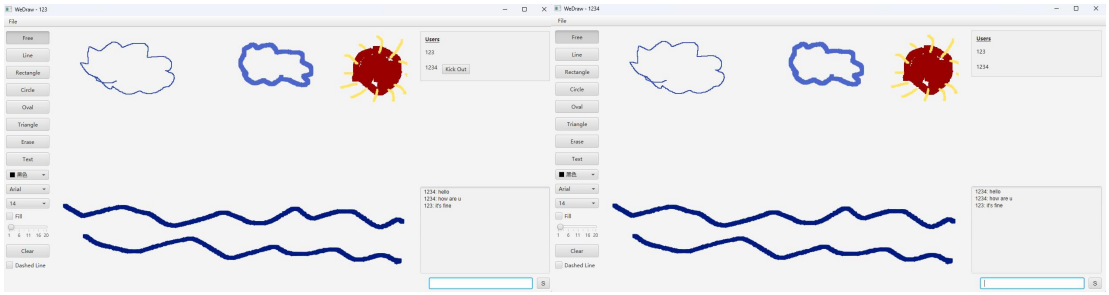
FileDropDown is a dropdown menu implemented using JavaFX and can only be operated by the Manager. The buttons in the menu are registered with click events. Drawing and chat commands are serialized and saved to a .wbd file using FileOutputStream and ObjectOutputStream. At the same time, the current whiteboard is converted into a Byte[] array and stored as an image. The method for reading files follows the same approach.

Full Name: Ninghao Zhu(Bruce) Student ID: 1446180 WeDraw

Screen Shots

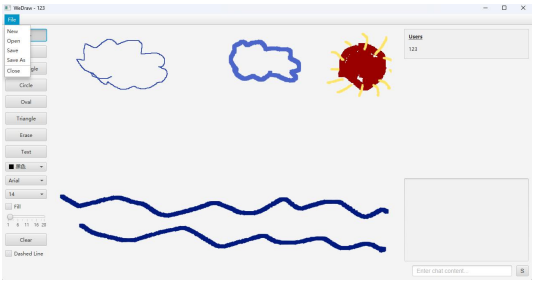


Login

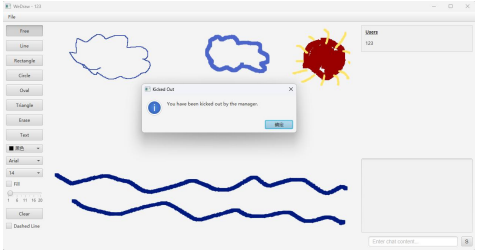


WhiteBoard

ChatPanel



FileMenu



Kickout

New Innovations

Whiteboard

The whiteboard allows users to customize colors using HSB, RGB, and Web formats. Users can also adjust hue, saturation, brightness, and opacity to create more diverse and expressive drawings. Additionally, users can modify text color, font size, and font style, providing a richer set of options for personalization.

Threads Worker Pool

The ThreadsWorkerPool has been optimized to better suit the CPU-intensive nature of the whiteboard application. The thread pool size is configured as the number of available CPU cores plus one, which minimizes the cost associated with creating and destroying threads. By utilizing a synchronized queue to manage concurrent tasks, this design improves scalability and optimizes the use of system resources.

Conclusion

WeDraw presents a fully functional and interactive distributed whiteboard system that supports real-time collaboration, drawing, chatting, and user/file management. By leveraging Java RMI and a multi-threaded architecture, the system ensures responsive communication and a smooth user experience. However, it currently lacks full serialization of the whiteboard state, which limits fast synchronization for newly joined users. Additionally, the FreeDraw and Erase modes are not yet optimized with dedicated threads, potentially causing performance issues during high-frequency drawing. Overall, the project effectively demonstrates core concepts of distributed systems in a practical and user-friendly environment, striking a balance between functionality and performance.