Multi-Threaded-Drawing-Board

Problem Context

In the contemporary world, digital collaborative tools have become an essential part of remote work and learning. This project aims to address the need for real-time collaborative tools by developing a distributed, multi-user shared whiteboard system where users can simultaneously draw and interact on a shared digital canvas.

Designing such a system presents various technical challenges, including handling concurrency, structuring the application and managing system state, ensuring efficient networked communication, and creating a user-friendly graphical interface.

The shared whiteboard system aims to provide both basic and advanced features. Basic features include the ability for users to draw shapes such as lines, circles, ovals, rectangles, and text in various colours. Advanced features include a text-based chat window for user communication, a file management system allowing new files, saving, loading and closing of the shared canvas, and user management capabilities allowing the manager to control user access.

The technology of choice for this project is Java's Remote Method Invocation, which allows objects to invoke methods of remote Java objects. The system will leverage multiple threads to handle different types of interactions, such as managing users and messages, tracking the state of the shared canvas, and responding to user actions on the UI. The RMI inherently handles thread initialization for each client that connects to the system, thus accommodating multiple simultaneous users.

In summary, the challenge tackled in this project is to design and implement a robust and efficient multi-user shared whiteboard system that can facilitate real-time collaboration while managing the complexities inherent in a distributed application.

Feature Analysis

Administrative Features

1. User Management: This feature permits managers on the server side to accept or reject client join requests. Additionally, if a user's behavior is inappropriate, the manager has the ability to remove them from the session.

2. File Management: The server manager has the capability to save, load, and save the current artwork in a JSON format. Importantly, the load feature provides global synchronization for all clients active in the current whiteboard session.

3. Canvas Creation/Clearance: This feature enables the server manager to delete or purge existing elements on the canvas.

Shared Features

1. Drawing: Users can draw various shapes and input text on the canvas, enabling collaborative artwork creation.

2. Chat: All participants in the current session have access to a chat room, fostering real-time communication among users.

Components of the System

The Distributed Shared Whiteboard application comprises several integral components that collectively ensure its efficient operation and deliver a seamless user experience.

RMI Package

1. IServer: This is the interface for the server-side RMI and is utilized for the RMI registry and lookup, enabling client access to its functions.

2. IServerImplementation: This class represents the implementation of the IServer interface. Its main responsibilities include maintaining shape lists, message lists, and client lists on the server-side, ensuring that UI components can be appropriately updated.

UI Package

1. WhiteboardUI: This class is the main UI interface handling both server and client UI local logic. It utilizes two additional threads for actively listening and monitoring changes in the server-side user list, shape list, and message list. This class is also responsible for updating the GUI for both the client and the manager.

2. EventListener: This helper class primarily implements the mouse and action listeners for the WhiteboardUI. It has a separate thread chiefly used to actively listen for any drawing action on the whiteboard and update the server's shape list accordingly.

3. RunServer/RunClient: These scripts are used to initialize server and client instances.

Utility/User Package

1. Client: This class is used to maintain client status, privileges, usernames, and other miscellaneous items.

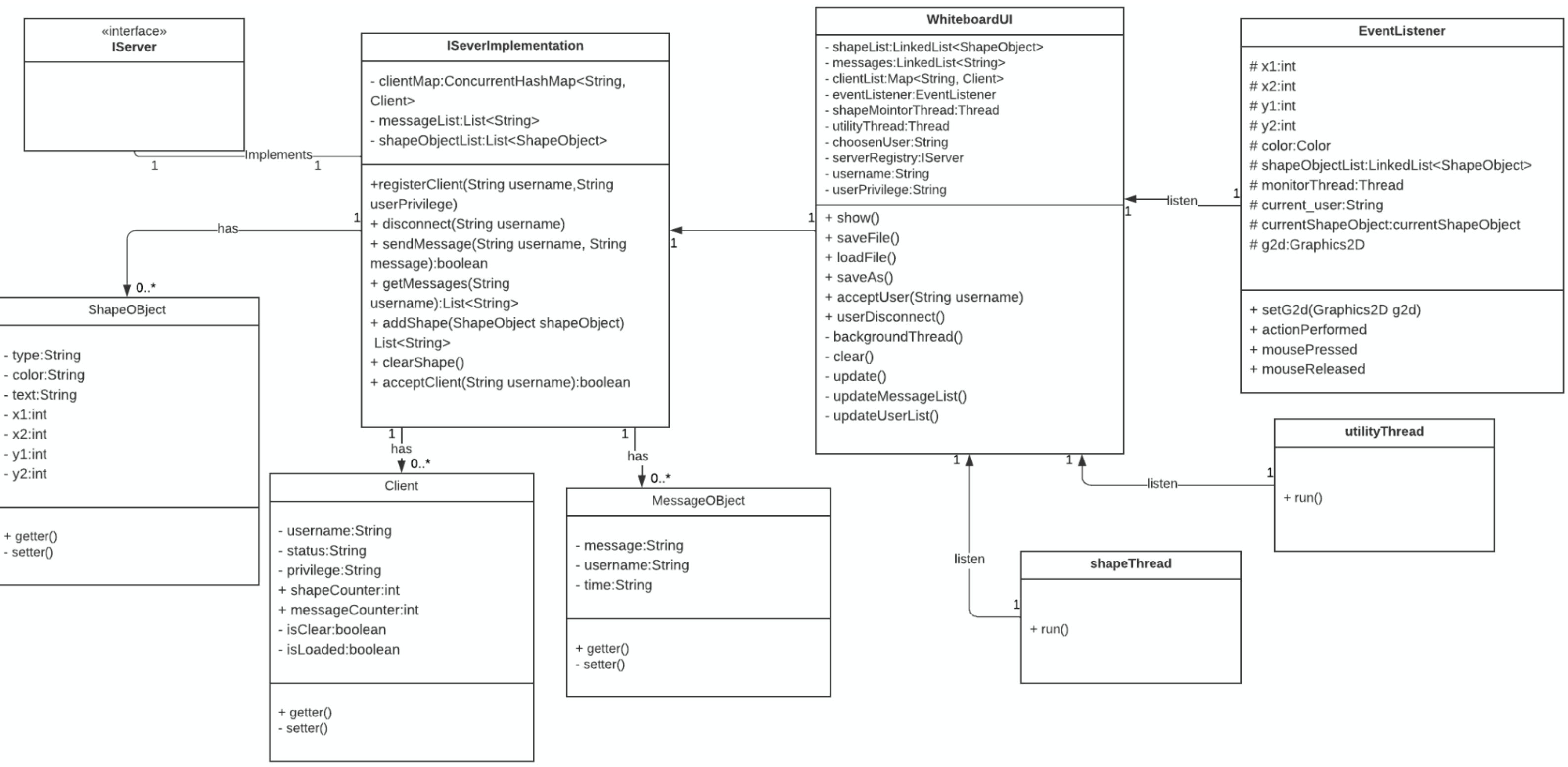
2. ShapeObject: This object captures the type of shape, its coordinates (x1, x2, y1, y2), and color properties. It can be directly converted to a JSON format using the Gson library function.

3. MessageObject: This object captures the message, username, and time properties. It too can be directly converted into a JSON format using the Gson library function.

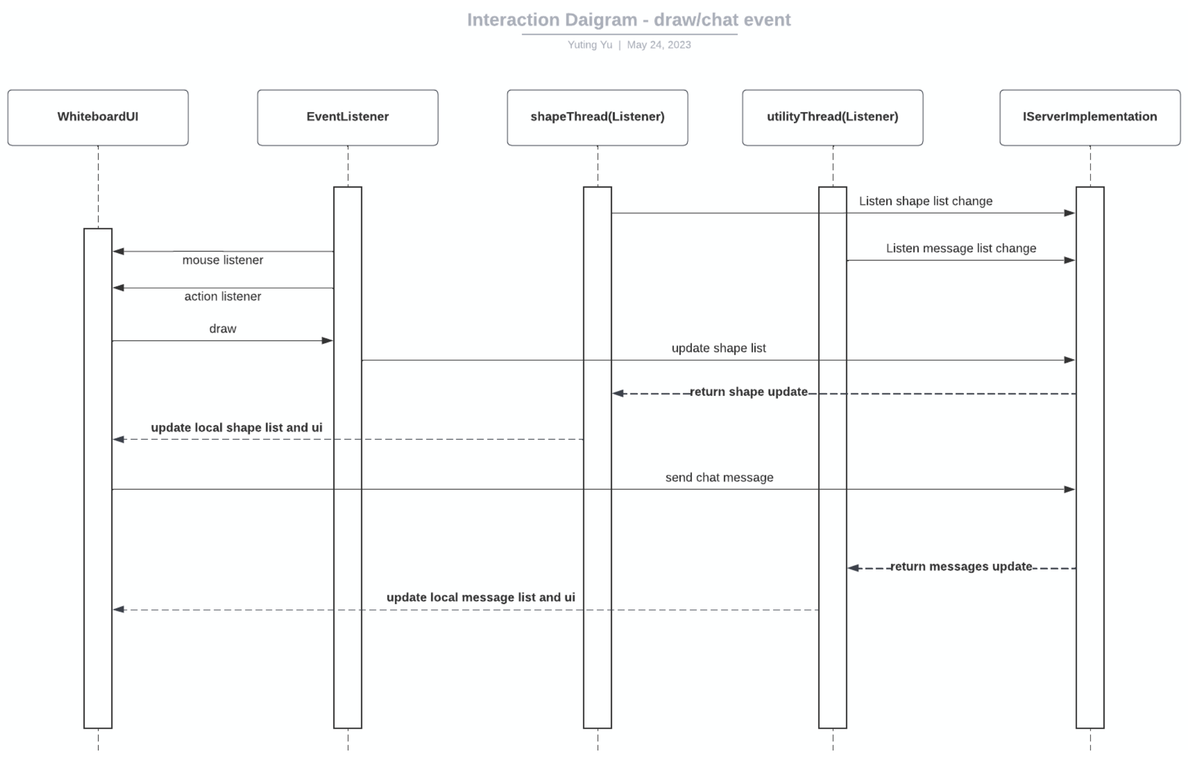
Each of these components plays a pivotal role in the operation of the Distributed Shared Whiteboard application, allowing for multi-user interaction, concurrency management, and a smooth overall user experience.

Class Diagram and Interaction Diagram

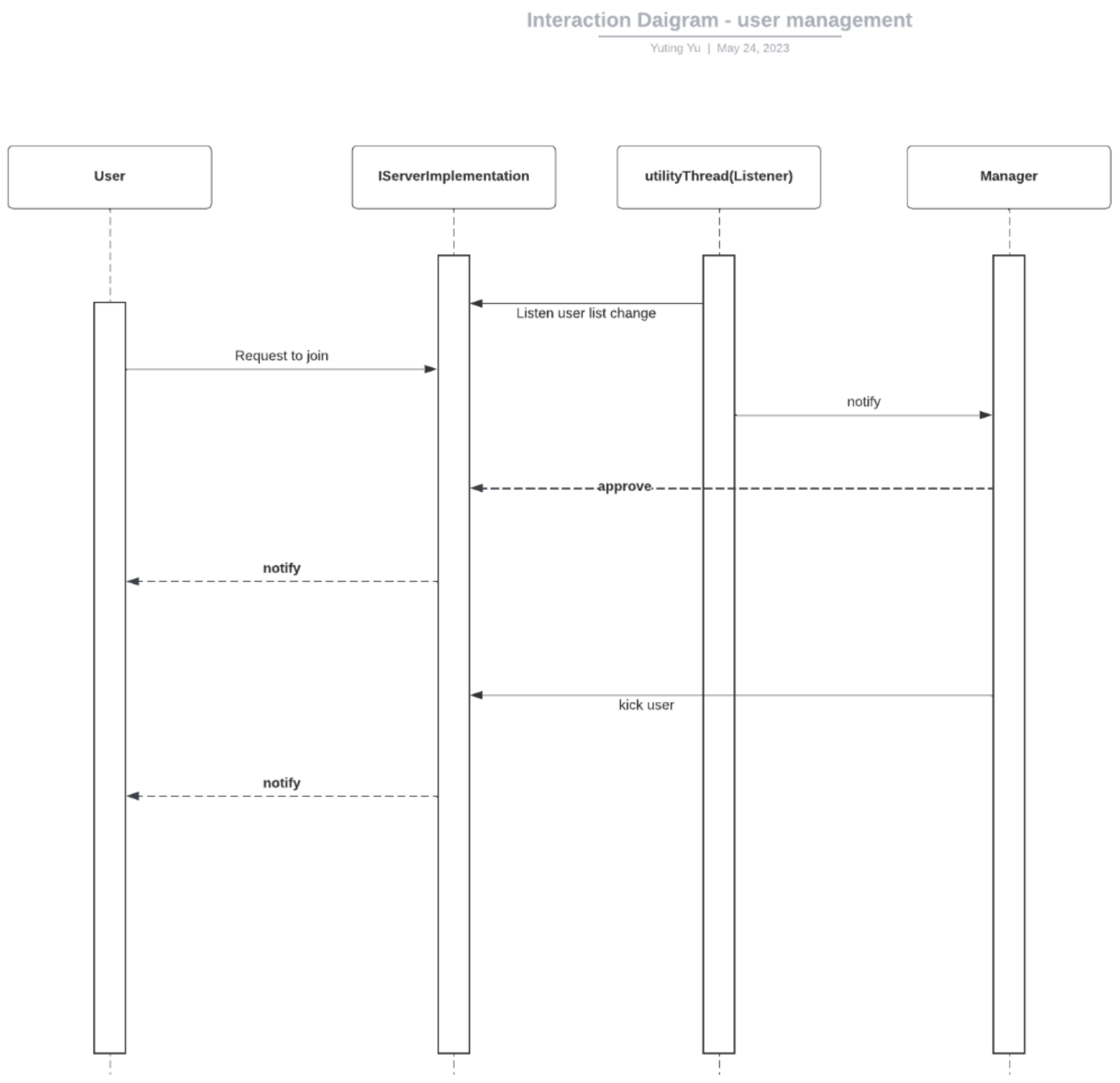
UML Diagram



Draw/chat function sequence diagram



User management function sequence diagram



Analysis

Architecture

Our system utilizes Java's Remote Method Invocation (RMI) to establish a robust and flexible connection between the client and the server. The inherent strength of RMI lies in its ability to seamlessly allocate a separate thread for each client request as per its internal implementation. This built-in feature ensures efficient handling of multiple concurrent client requests without requiring explicit thread management in our system.

Supplementing this core RMI functionality, we initialize three additional threads per user and client. This design choice supports our goal of maintaining real-time interactivity and responsiveness in our shared whiteboard application, irrespective of the number of active users.

The first two of these additional threads manage drawing actions. One thread, which originates from the EventListener class, is dedicated to tracking mouse movements on the canvas. This thread communicates the latest drawing shapes to the RMI server, enabling real-time sharing of drawings. The second thread, the 'ShapeThread', is designed to actively listen for changes in the shape list on the RMI server. It ensures that any modifications made by any user are immediately reflected on the local UI of all active users.

The third additional thread, the 'UtilityThread', monitors user join/leave operations and message transmissions. This thread is essential for maintaining a dynamic user list and supporting real-time text-based communication among users.

This architecture capitalizes on the strengths of RMI's automatic thread allocation while augmenting it with additional threads to cater to our specific application needs. The design offers several advantages:

1. Concurrency Management: By allocating separate threads to handle different user actions, our design effectively manages concurrency, ensuring smooth operation even with multiple active users.

2. Real-Time Interactivity: The dedicated threads for tracking drawing actions and updating the local UI ensure that all users can view and contribute to the shared canvas in real-time.

3. User Management & Communication:** The UtilityThread enables dynamic user management and real-time communication, enhancing the collaborative experience.

4. Scalability: With the combination of RMI's thread allocation and our additional threads, the system can scale to handle an increasing number of users without compromising performance or interactivity.

The selection of this design embodies a commitment to efficient resource management, performance, and a high-quality user experience. The architectural choice ensures that our shared whiteboard application is interactive, responsive, and capable of supporting collaborative work in a distributed environment.

Concurrency

In the architecture of this distributed shared whiteboard system, managing concurrency is crucial. Java's Remote Method Invocation and additional multi-threading are employed to handle concurrent interactions.

First, Java's RMI naturally supports concurrent client requests. RMI allocates a new thread for every incoming request, effectively balancing load and facilitating multiple simultaneous user interactions.

Second, to complement RMI's concurrent processing, additional dedicated threads are implemented for each user. Specifically:

1. EventListener Thread: This thread tracks users' drawing actions on the canvas and updates the server in real-time. This design supports simultaneous drawing by multiple users on the shared canvas.

2. ShapeThread: This thread actively listens for changes to the server's shape list. As users add or modify shapes on the canvas, the server's shape list is updated. The ShapeThread then updates all client interfaces to reflect these changes.

3. UtilityThread: This thread monitors user join/leave operations and message exchanges. It ensures smooth chat operation and user status updates, enabling real-time interaction among users.

These three threads work in conjunction to ensure a seamless user experience even with multiple simultaneous interactions. However, attention to data consistency and synchronization is essential, particularly when dealing with shared resources such as the shape list. This design considers these issues to handle concurrent modifications effectively.

Communication Protocol

In this distributed shared whiteboard system, the communication protocol plays an essential role in orchestrating interactions between the client and the server. The system employs Java's Remote Method Invocation as the primary communication protocol.

RMI is a Java API that performs the object-oriented equivalent of remote procedure calls , with support for direct transfer of serialized Java classes and distributed garbage-collection.

Message Format

The message format adopted for this project is a JSON string. This decision is guided by the inherent support for strings provided by RMI, eliminating the need for additional modifications on class objects for serialization. Moreover, JSON objects streamline the implementation and maintenance of the save/load feature. All saved whiteboard files are stored in JSON format, a human-readable format that enhances the clarity and accessibility of the data.

Conclusion

In conclusion, the Distributed Shared Whiteboard project aims to offer a real-time, multi-user, collaborative environment built using Java and RMI. The system caters to diverse drawing needs, along with advanced features like chat and file management, thereby promoting seamless remote collaboration.

The architecture exploits multiple threads and RMI's inherent thread handling capabilities to address concurrent user actions effectively. Concurrent actions are managed using Java's Concurrent LinkedQueue, ensuring synchronization and avoiding data inconsistencies.

The system employs RMI for remote communication, and JSON strings are used for message formats, aiding in serialization and enhancing human readability of saved files.

Through careful design and attention to concurrency, communication protocols, and data structures, this project strives to deliver an efficient, robust, and user-friendly shared whiteboard platform that meets the demands of modern collaborative workspaces.