

SWEN90015 - Distributed Systems

Report for Assignment 2

Lai Wei Hong
ID:1226091

University of Melbourne
e-mail: weihong@student.unimelb.edu.au

May 25, 2021

Contents

| | | |
|-------|--|----|
| 1 | Introduction | 3 |
| 2 | Architecture | 3 |
| 2.1 | Overall Architecture | 3 |
| 2.2 | Client UI Overview | 3 |
| 2.3 | Communication Protocol | 5 |
| 2.3.1 | Communication Protocol: Document Object and Shape Objects | 5 |
| 2.3.2 | Communication Protocol: Free Drawing | 5 |
| 2.4 | Key functionality | 7 |
| 2.4.1 | Meeting room creation | 7 |
| 2.4.2 | Joining an existing meeting room | 7 |
| 2.4.3 | Whiteboard initialization | 7 |
| 2.4.4 | Whiteboard updates | 7 |
| 2.4.5 | Chatbox updates | 8 |
| 2.4.6 | File operations | 8 |
| 2.4.7 | User exiting and session termination | 8 |
| 3 | Failure Models | 10 |
| 3.1 | Server | 10 |
| 3.1.1 | Fault 1: Client cannot be reached during an update operation | 10 |
| 3.1.2 | Fault 2: Number of rooms exceed the initialized limit | 10 |
| 3.2 | Client | 10 |
| 3.2.1 | Fault 1: The client cannot read the file chosen by the user | 10 |
| 3.2.2 | Fault 2: The server is not reachable by the client | 11 |
| 4 | Conclusion | 11 |

1. Introduction

In this report, we will discuss the implementation details and design decisions that went into implementing the distributed whiteboard application. In the first section, we will talk about the overall system architecture and the behaviour that the system exhibits throughout the course of its operation. Following that, we will discuss some of the key fault models that might affect this system. In the final section, we will introduce some possible improvements that can be made to the system and give a general overview of the achievements that have been made in this assignment.

2. Architecture

This section details the overall architecture of the distributed whiteboard system.

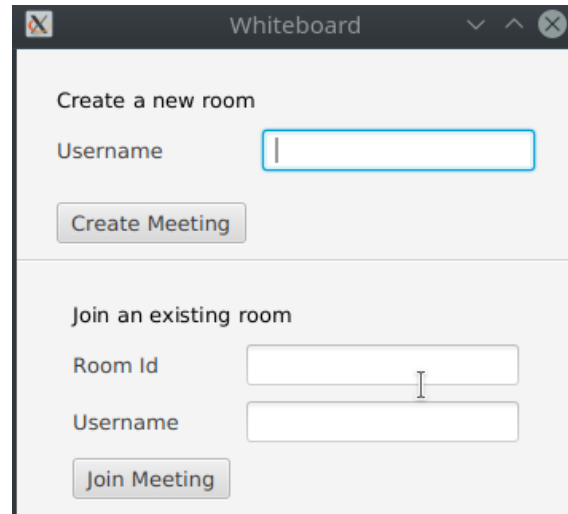
2.1. Overall Architecture

The system consists of two major parts, the server, and the clients. The clients are the user-facing application that enables the user to interact with the whiteboard and the messaging service. The server, on the other hand, serves to synchronize the state of the shared whiteboard and message board among all users within the same room. The client and server interacts with each other to exchange resources.

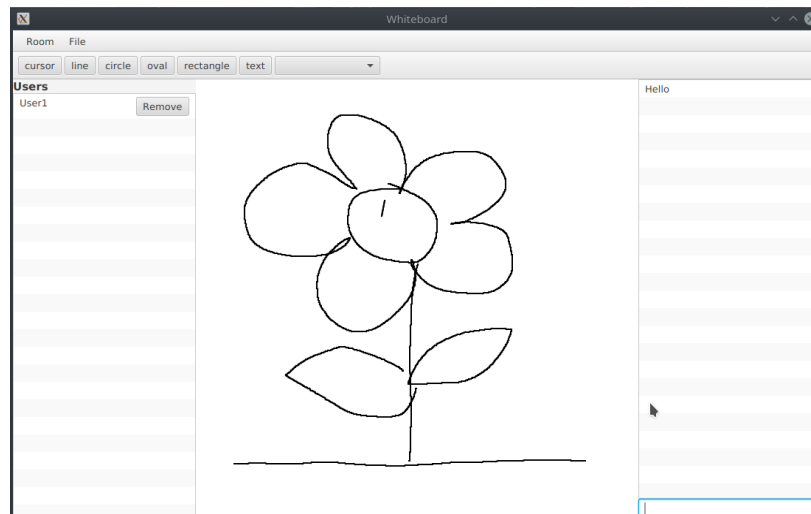
A room in our system is defined as an entity within the server where all the meta-data for the application is stored. It contains the shared state of the whiteboard as well as the message logs for a whiteboard session. In the following sections, a shared whiteboard session will be referenced as a room.

2.2. Client UI Overview

The client in our system presents two different views during operation. The initial view consists of a menu where users can choose to create a meeting room or join an existing room. Only when a user successfully joins a room will they be able to access the secondary view. The secondary view consists of a list of users in the session on the left, a whiteboard in the middle, and a message box on the right. The item menu for file operations is disabled for non-managerial users. The same applies to the button to kick users from the session.



The image shows a window titled "Whiteboard" with a standard macOS-style title bar (red, yellow, and green buttons). The window is divided into two main sections. The top section is titled "Create a new room" and contains a "Username" label followed by a text input field with a blue border. Below this is a "Create Meeting" button. The bottom section is titled "Join an existing room" and contains two labels, "Room Id" and "Username", each followed by a text input field. Below these fields is a "Join Meeting" button.

Figure 1: View 1**Figure 2:** View 2

2.3. Communication Protocol

The Remote Method Invocation (RMI) API provided by Java serves as the basis of the communication protocol in this system. All messages passed between the server and the client utilises the RMI. This simplifies the development process of the distributed whiteboard application as it removes the need to handle the low-level details present in HTTP/ TCP-based approaches.

In general, two interfaces were defined for this application. The first one is the MeetingRoom-Service interface. The implementation of this interface is hosted on the server-side and is called by the client when it needs to access resources held within the server. The second one is the Client-Callback interface. This implementation of this interface is done in the client-side and servers to called by the server to propagate update operations to the user when there is a change in the shared whiteboard state.

Since not all objects are serializable, the transmission of shape data and continuous lines are handled differently.

2.3.1. Communication Protocol: Document Object and Shape Objects

Completed shapes such as circles, texts and rectangles are packaged as shape objects. Before transmission, this objects are transformed into JSON string using the GSON library. Since Strings are serializable, this approach enables the objects to be transmitted using the RMI API. Once the recipients receive the JSON string, the object can be retrieved by decoding the JSON string.

There are situations where the entire state of the whiteboard needs to be transmitted. In this case, a Document object is used. The document object stores information about the three types of shapes that can be drawn in the whiteboard, ie text, shapes and freeshapes. The transmission method is similar to that mentioned above.

2.3.2. Communication Protocol: Free Drawing

A free drawing is an action where the user draws without using any of the predefined shapes. It would not be wise to store every pixel that the user have drawn individually as it would make retrieval of other shapes within the whiteboard to be computationally expensive.

To deal with this, the pixels that the cursor draws when in free drawing mode is merely replayed and not stored by the server. When the user releases the mouse button, signalling the completion of the drawing, the client creates a FreeShape object that contains an approximation of the points that are relevant in drawing the free shape. Similar to any other shape objects, the FreeShape object is transmitted to the server to be stored.

```

import java.rmi.Remote;
import java.rmi.RemoteException;
import java.util.List;

public interface MeetingRoomService extends Remote {
    String createNewRoom(String userId) throws RemoteException;
    boolean joinRoom(String roomId, String userId) throws RemoteException;
    boolean exitRoom(String roomId, String userId) throws RemoteException;
    boolean kickUser(String roomId, String userId, String kickerId) throws RemoteException;
    boolean sendMessage(String msg, String roomId, String userId) throws RemoteException;
    boolean addShape(String s, String roomId, String userId) throws RemoteException;
    boolean addFreeLines(String s, String roomId, String userId) throws RemoteException;
    boolean addFreeShape(List<String> list, String roomId, String userId) throws RemoteException;
    boolean addText(String str, String roomId, String userId) throws RemoteException;
    boolean clearDoc(String roomId, String userId) throws RemoteException;
    boolean setInitialisationStatus(String roomId, String userId) throws RemoteException;
    boolean setDocument(String roomId, String userId, String docJson) throws RemoteException;
    String getDoc(String roomId, String userId) throws RemoteException;
    int getUserRole(String roomId, String userId) throws RemoteException;
    List<String> getUserList(String roomId, String userId) throws RemoteException;
    List<String> initFreeShapesDoc(String roomId, String userId) throws RemoteException;
    List<String> initShapesDoc(String roomId, String userId) throws RemoteException;
    List<String> initTextDoc(String roomId, String userId) throws RemoteException;
}

```

Figure 3: MeetingRoomService

```

import java.rmi.Remote;
import java.rmi.RemoteException;
import java.util.List;

public interface ClientCallbackService extends Remote {
    void updateMessage(String message) throws RemoteException;
    void updateShapes(String shapeJson) throws RemoteException;
    void updateText(String textJson) throws RemoteException;
    void updateUserList(List<String> userId) throws RemoteException;
    void removeUser() throws RemoteException;
    void createnewDocument() throws RemoteException;
    void updateDocument(String docJson) throws RemoteException;
}

```

Figure 4: ClientCallbackService

2.4. Key functionality

In this section, we will detail how the interactions between the different components in the system result in the system achieving the functionality outlined in the specifications. The diagram below shows the order of the actions that might take place throughout the operation of a meeting room. Attached below is the lifecycle of a room in the shared whiteboard system.

2.4.1. Meeting room creation

When a user creates a new room through the client, the client calls the `newMeetingRoom` function via RMI to the server. The server loads the resources required by the room and sends a confirmation back to the user. Upon receiving the confirmation, the client loads the second view to give the user access to the whiteboard, If the server capacity is reached, a rejection is sent to the user and an alert box will pop up informing the user that the operation has failed.

2.4.2. Joining an existing meeting room

When a user joins a new room, he/ she needs to supply a valid room number and username at the primary view of the client. The client interfaces with the server to check the validity of the room number and username. If either one of the parameters fail the check, an alert box will pop up to inform of the failed operation. Otherwise, the client will load the secondary view for the user.

2.4.3. Whiteboard initialization

When a user joins the room, the client interfaces with the server to retrieve the current state of the whiteboard as well as the list of user currently in the room. The server will return a document object transformed as a JSON string using the GSON library. The Document object stores lists of the three components within the whiteboard, ie shapes, free shapes and texts. The client parses the document object and update the whiteboard at the client-side to reflect the current state of the shared whiteboard.

2.4.4. Whiteboard updates

When a user makes any changes to the whiteboard (adding new shapes or texts), the client sends an update to the server. Once the server has updated the stated of the whiteboard, it spawns a new thread to propagate changes to all the users within the meeting room, This enables all theusers to see the updates made by one user. Similar to the whiteboard initialization phase, the Shape objects are transformed into JSON string as the Shape objects are not serializable.

2.4.5. Chatbox updates

Unlike the whiteboard, the chatbox does not have an initialisation process. This decision is made due to the overhead that comes with distributing and synchronising long lists of texts across multiples users.

The chatbox, however, does have an update function that is similar to that of the whiteboard. Any new messages that were sent to the server, which process and distribute the text to all the other users in the room.

2.4.6. File operations

File operations are enabled only to the manager, so it will be greyed out for other users. Three file operations were defined for this assignment, they are new file creation, file opening and save file as. Since using anyone of these operations alter the state of the managers whiteboard, the same update rules mentioned above apply here. Given that we have gone in depth in the section above about whiteboard state updates, we will be focusing on the file operations in this section.

The file creation operation simply replaces the current state of the whiteboard with a blank state. Upon completion, the manager sends an empty document to the sever to update its state.

When a user opens a file, the client uses the FileChooser functionality provided by JavaFX to create a reference to the target file. Once the target file is identified, it is passed through a parser to extract information about the saved whiteboard. If no errors occur at the end of reading operation, the client updates the whiteboard with the saved whiteboard and sends the update to the server.

Similarly, the save file operation uses the FileChooser functionality to choose the target file to save the whiteboard. However, in an effort to keep the application as lightweight as possible, the meta-information of the whiteboard is not maintained client-side. So, before saving the file, the client sends a request to the client to get a copy of the Document object that houses all the information of the shapes in the whiteboard. With the use of GSON, the Document object is encoded into a JSON file.

2.4.7. User exiting and session termination

There are two scenarios that can occur when a user leaves a room. One is when the manager leaves the room. In this case, the server terminates the room and informs all the users in the room about the session termination using an alert box. Following that, all the users will be shown the initial view of the client.

A second scenario would be when a non-managerial user exits the room. In this case, the server updates the state of its user list and sends an update to all the users in the room about the changes to the user list.

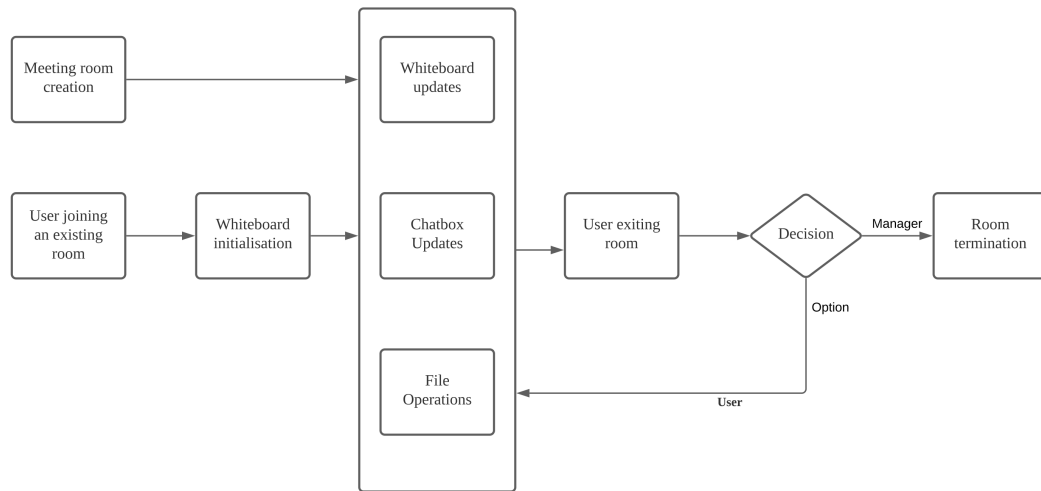


Figure 5: Lifecycle of a room in the shared whiteboard system

3. Failure Models

In this section, we will discuss about different kinds of faults encountered by the system and actions taken to handle it. As mentioned in the assignment brief, all communications are assumed to be reliable, so these faults will not be considered.

- Data packet loss
- Data packet manipulation/Changes in data packets due to noise

3.1. Server

3.1.1. Fault 1: Client cannot be reached during an update operation

- Overview
The client may not be reachable during the update operation. This can be due to the client undergoing the initialization process and is not yet ready to receive updates from the server yet.
- Error correction
When a user finishes drawing, the server synchronizes the entire whiteboard state with all the users. This ensures that the clients that have missed an update, eventually receive it. This enforces the liveness property for the system.

3.1.2. Fault 2: Number of rooms exceed the initialized limit

- Overview
There is a hard-coded limit to the number of rooms that can be opened to ensure the stability of the system. So, when a user attempts to create a new room when the limit has been reached, errors will be thrown in the server-side as it ran out of resources to create a new meeting room.
- Error correction
The server, through the ClientCallback interface, updates the client on the failure of room creation. Upon receiving the update, the client displays an alert box to inform the user that the room cannot be created.

3.2. Client

3.2.1. Fault 1: The client cannot read the file chosen by the user

- Overview
The manager may be opening a file that has an incorrect format or is corrupted. This can lead to errors in the parser when the client is reading the file.
- Error correction
Any errors thrown by the parser are caught by the application. The client will create an alert box to inform the user that the reading action was invalid.

3.2.2. Fault 2: The server is not reachable by the client

- Overview

The server may not be running when the client sends a request. This could lead to the client throwing `NotBoundExceptions` or `RemoteExceptions`.

- Error correction

Similar to fault 1, we catch the exceptions thrown by the client and display it in an alert box. This informs the user of the error which enable them to act accordingly, either by retrying later or connecting the system administrators on the root cause of the disruption.

4. Conclusion

Overall, this development of the distributed whiteboard system has been a success. It has given me a deeper understanding of working with the RMI API. All the desired functionalities outlined in the specifications were implemented in the final version of the system.

However, multiple improvements can be introduced to the system to increase the usability and functionality of the system.

- Elect a new manager to the session when the current manager leaves
- Extend the system to working with multiple servers concurrently
- Extend the system to work across the internet. (May be unrealistic due to issues surrounding firewalls)