COMP90015 Distributed System

Assignment 2 - Report

Jiaxu Kang 1219581

Yuhao Wang 1208524

Problem Analysis

Distributed shared white board is an application/tool to allow multiple users communicating with each other by drawing and sharing ideas. It requires operations to render immediately to other users including drawing shapes, changing colors, texting and etc. In order to achieve this, a distributed system(server) needs to be implemented to handle all the interactions. Additionally, each white board instance is managed and owned by one individual user. Any exceptions like rejection, server not found and etc should be handled properly by the system in case of system crash. The main goal is the information sharing in real time, thus a good solution to it is to build a server/manager to manipulate all the top-level operations including create a white board, save a white board, kick some users out of the white board, and multiple users at the bottom-level who are granted authority to write, draw and text. Lastly, a well designed GUI should be implemented as well to let users login and quit the white board.

Feature Introduction

Basic Features

A distributed system is created to allow multiple users to draw on a white board and the information is sent to the rest simultaneously.

Only one manager(the first user) can own the white board.

Any joined users(user list) are seen by the rest.

The system can handle any kinds of exceptions like invalid port number, server not found and failed connection.

A login GUI is implemented to read port number and name inputs.

A white board GUI is implemented to let users choose from line, pen, circle, triangle, rectangle, text and color set.

Advanced Features

The manager can save any information on the white board as a png and output a message saying “Save Successful!”.

Users can clear the white board by clicking on a specific button.

A chat section is implemented for all users to communicate and share ideas. Each time an user finished his/her sentence and then click on the enter button, a text including his/her name and the sentence will be sent to the text window so that everyone else can see.

The manager can kick any user out of the white board if someone does not follow instructions or rules and the manager will receive a message indicating if someone is kicked out.

Loading any previous images is also implemented in the system that the manager has the right to open files.

When a new user tries to join, the manager will receive a request first and make a decision to let this user get in or not. If manager is disconnected, a corresponding message will display. And a previously kicked user cannot join the white board again.

Code Structure

Bottom Level

We describe message, position and tool as a section of model that each instance will be used later.

Message class simply implements Serializable that has two private and final attributes(name and next), which are Strings. Two getters are used to retrieve the information stored in Message. Each time, users send a text message in the chat room, one corresponding Message will be created.

Position class also implements Serializable that has private and final integer attributes x and y to store the exact location for mouse clicking. Two getters are included.

Tool class just regulates those user operations including line, pen, circle, triangle, rectangle and text as a enum type. It is used to find which operation an user is trying to conduct.

IRemoteWhiteBoard class extends Remote class and inherits many useful methods(details see in the IRemoteWhiteBoard.java) like acceptUser and RefuseUser.

Top Level

In this level, there two main parts, which are Client and Server. The server initiates the whole connection and the white board application. Each Client is individual user who is trying to interact and communicate.

Client class is consisted of a String indicating host's name, a int port number that holds which port it is connected to, a String name of its user, a Registry imported and an instance of IRemoteWhiteBoard. Corresponding getter methods are implemented as well.

On the server side, IRemoteWhiteBoardlmpl is a class that extends UnicastRemoteObject and implements IRemoteWhiteBoard. It stores all the essential data ensuring the system works

properly. All the lists like clients, waitList and refuseList are designed to be synchronized since we are doing concurrent programming that thread-safe needs to be guaranteed. Two hash maps(textInfo and whiteBoardContent) are using concurrentHashMap for the same purpose. The methods override class definitions implemented from IRemoteWhiteBoard. Most of which are clear and precise to understand, so I will introduce a few of them that are more related to our white board concept. In the userNameAlreadyExist method, each user is identified by its user name which means each name is unique and users must choose another name in order to login if the previous one is chosen. In the reset method, all previously stored information will be erased. WhiteBoardContent and textInfo will be initialized from the beginning, and the preDrawImage will be cleared as well. When the manager chooses to open a file(.png), the preDrawImage will be converted into bytes and stored in a byte list for later use.

The Server class has a default port which is 3000(could be changed if it is occupied or a specific port number is entered). Then, it will check the validity of the port number and output an error message if it is incorrect. If the process goes through successfully, a remoteWhiteBoard class wil be initialized and being bound to the Registry.

UI Level

A window for login is structured here by using swing. Each button, label and text fields are designed in a user-friendly manner. Users can enter their user names in a specified field. A

try-catch block in the action Performed method will catch any exceptions like invalid port number preventing from system crash.

After an user is logged in, a white board UI will be displayed. Anyone could scroll the user list to see who has joined the white board. All the essential tools including Line, Pen, Circle, Triangle, Rectangle, Text, and Color set are initialized by the use of JButton so that users could click on them and do the operations. Each mouse click will be read as inputs, so that the system would know which location the user wants to draw his/her circle on.

OpenServer simply creates a server and OpenClient class initiates each thread using Runnable and render the LoginPage window to each of them.

Conclusion

Overall, the implementation was a success that accomplished all the fundamental features as expected and no major errors occurred during running the code except a small lag happened sometimes when sharing drawing information. The system successfully constructed a platform for users to interact by drawing and texting concurrently. For future improvements, more tools could be added to the option lists, like changing fonts and size so that users could write articles or essays together instead of just a simple draft. For the drawing part, color filling in certain shapes is also feasible that makes the image more vivid. And the clear function should be replaced by an eraser and new page functions in case users want to erase a small portion of the whole instead of clearing everything and starting from the beginning.