COMP90015: Collaborative Whiteboard

Kasper Terndrup, 918581, kterndrup@student.unimelb.edu.au; Marc Tom Thorgersen, 906200, mthorgersen@student.unimelb.edu.au; Navnita Nandakumar, 921834, nnandakumar@student.unimelb.edu.au; Xin Qi, 856094, xingq3@student.unimelb.edu.au

October 17th 2017

Introduction

In this project, we are tasked with the job of creating a collaborative whiteboard that can be shared between clients. A collaborative whiteboard allows multiple users to draw simultaneously on a canvas. With the inclusion of a simple chat window that allows all the current users of the system to broadcast messages to each other, it makes for a convenient tool to aid in collaborative discussions.

I.I Requirements

GUI

The whiteboard needs to support the usage of Shapes (line, circle, rectangle and oval), Free draw, Text and Erase in a minimum of 16 colours. A text-based Chat Window should also be implemented, which allows communication between users. A unique whiteboard manager should be able to access a file menu with options to create a new file, save (as well as "save as"), open a previously saved filed, and close the existing file.

Extensions: Our version supports the movement of text, additional colours and stroke sizes. We also included an Edit menu with options for the Manager to Undo, Redo and Erase the board.

Clients

The users are required to provide a username when joining a whiteboard. The interface must show the usernames of all other clients currently editing the whiteboard; this online peer list must be maintained. Clients may connect/disconnect at any time.

The user who creates the whiteboard becomes its Manager. Other users may request to join the whiteboard by providing the Manager's IP address and Port number as inputs, which must be approved by the Manager. The Manager may also kick someone out, at which point the user in question must be notified.

When the Manager quits, the application will be terminated and all other peers will be notified.

System Architecture

Our system includes a client/server architectural model that allows the clients and server to be hosted on different systems in the network.

II.I Server

The server is responsible for managing the clients currently connected to it and for the distribution of the whiteboard drawing and chat messages. The server also listens for new incoming connections.

Manager

In our implementation, the first user to create the whiteboard (and therefore its Manager) acts as the Host for the application. The manager may create a New whiteboard file, Save (and Save As) and Close the existing file or Open a previously created file. The manager also has the privilege to kick out another user and edit the whiteboard (Undo,Redo and Erase Board operations). The Manager approves/rejects requests by other users to join the whiteboard.

II.II Client

The other users that request to join the whiteboard are the clients. They must place a request to the Manager to join the whiteboard. Clients may Quit or Join the whiteboard at any time and use all whiteboard features. However, they cannot use the File and Edit options.

Communication Protocol

III.I RMI

We used the Java RMI (Remote Invocation Method) as the communication protocol in our project since it supports the direct transfer of serialized Java classes and distributed garbage-collection (using DGC).

Server

The Server class listens to the RMI requests and implements the interface where the methods are declared.

ServerI

The ServerI interface defines the interface implemented by the Server and contains the declarations of the methods to Register/Unregister the Client and Broadcast the message and Update the canvas.

Client

The Client class gets the reference to the remote object residing on the server and invokes its broadcast and update methods.

ClientI

The ClientI interface defines the interface implemented by the Client and contains the declaration of the method to Retrieve the message and Update the canvas.

Implementation

IV.I GUI

The GUI (Graphic User Interface) was implemented using AWT and Swing components.

AWT (Abstract Window Toolkit) is a set of APIs (Application Program Interfaces) that provides a GUI for a Java program. The drawback is that it is platform-dependent.

Swing is a GUI widget toolkit for Java and part of the JFC (Java Foundation Classes) API for providing a GUI for a Java program. It provides a more sophisticated set of GUI components than AWT and a native look and feel that emulates the look and feel of several platforms. It also supports a pluggable look and feel that allows applications to have a look and feel unrelated to the underlying platform. Swing components are therefore "lightweight" and more powerful and flexible than AWT equivalents.

IV.II WhiteBoard Classes

The main Whiteboard contains 8 classes, introduced below by the order of their appearance:

WelcomeWindow

A welcome dialog is presented using the InputAddrWindow class, which requires the user to choose whether he/she wishes to "Join an existing whiteboard" (act as a client) or "Create a new whiteboard" (act as the host).

If the user chooses to be a client, then he/she must input the IP address and Port number the Manager used to create the whiteboard. If the Manager approves the user's request to join, he/she is enlisted as a peer.

If the user chooses to be a host , then his/her IP address and specified Port number becomes the server address other peers need to input when requesting a connection.

WhiteBoardGUI

This class is used for presenting the main GUI of the whiteboard. It contains three main panels, namely, Chat History, Drawing Board and Function panel.

The Chat History panel is used to present the chat history and display the list of online users. The manager will also have options like "Kick out User" in this panel (that other users will not).

The Drawing Board is the panel where the actual whiteboard functionality takes place. Users can draw, erase and type and move text within this panel. This panel invokes the DrawBoard class (detailed later).

The Function panel presents the colour choices and drawing options. It has 16 default colours, but the user may choose other colours by clicking the "More Colors" button. The drawing options include shapes like Line, Circle, Rectangle and Oval, as well as Text, Choose (to move the Text) and Erase functions. The "Choose Stroke" button can be used to set the size of the stroke.

The main window also includes two menus- File and Edit. However, options like New File, Open, Save, Save As and Close from the File menu, and the entire Edit menu (with options like Undo, Redo and Erase Board) are only visible to the Manager. Other users can only choose to "Quit" the whiteboard from the File menu.

Note: It might be helpful to note the distinction between the Erase Board and Close/New File functions. Erase Board option allows the Manager to erase all diagrams from the canvas of the existing whiteboard. The New File function allows the Manager to create a new whiteboard with a different Port number (but same IP address), while the Close function allows the Manager to close/exit the existing whiteboard.

DrawBoard

This class is mainly responsible for performing the drawing actions. The shapes are made using the Java2D shapes method and each shape is a separate object with properties like X- and Y- co-ordinates, Height and Width. A ColoredShape class (detailed later). is used to store these properties.

We implement the MouseListener to acquire the shape's properties.

ColoredShape

This class represents an abstraction over any change that a user can make to the whiteboard. This means that the class holds any figure, text, or erase-action that a user can make. This allowed us to simplify the process of updating the DrawBoards across mulitple clients, such that only one type needed to be drawn.

User

This class is used to store user properties like IP address, Port number, Username and Status (Host/Client).

FileUtil

This class defines several file-related methods such as Save, Open and Export to Image.

MyWindowListener

This class is used to define the window actions. For example, when the Manager tries to close the window, a dialog box appears asking whether he/she would like to save the file first.

IV.III Chat Functionality

The chat functionality in the system works exclusively through the RMI implementation. By utilising the RMI registry method `broadcastMessage`, a client asks the server to broadcast a given message to everyone, including the client itself. The `broadcastMessage` method calls the `retrieveMessage` method on each client, a function which retrieves the message the server was asked to broadcast.

IV.IV RMI

The RMI components are split into two packages, one for server and one for client. Each package contains two classes, a driver and a base class, and a interface. The driver classes are solely responsible for establishing connections. In the case of the server this means creating the registry, and in the case of the client connecting to said registry.

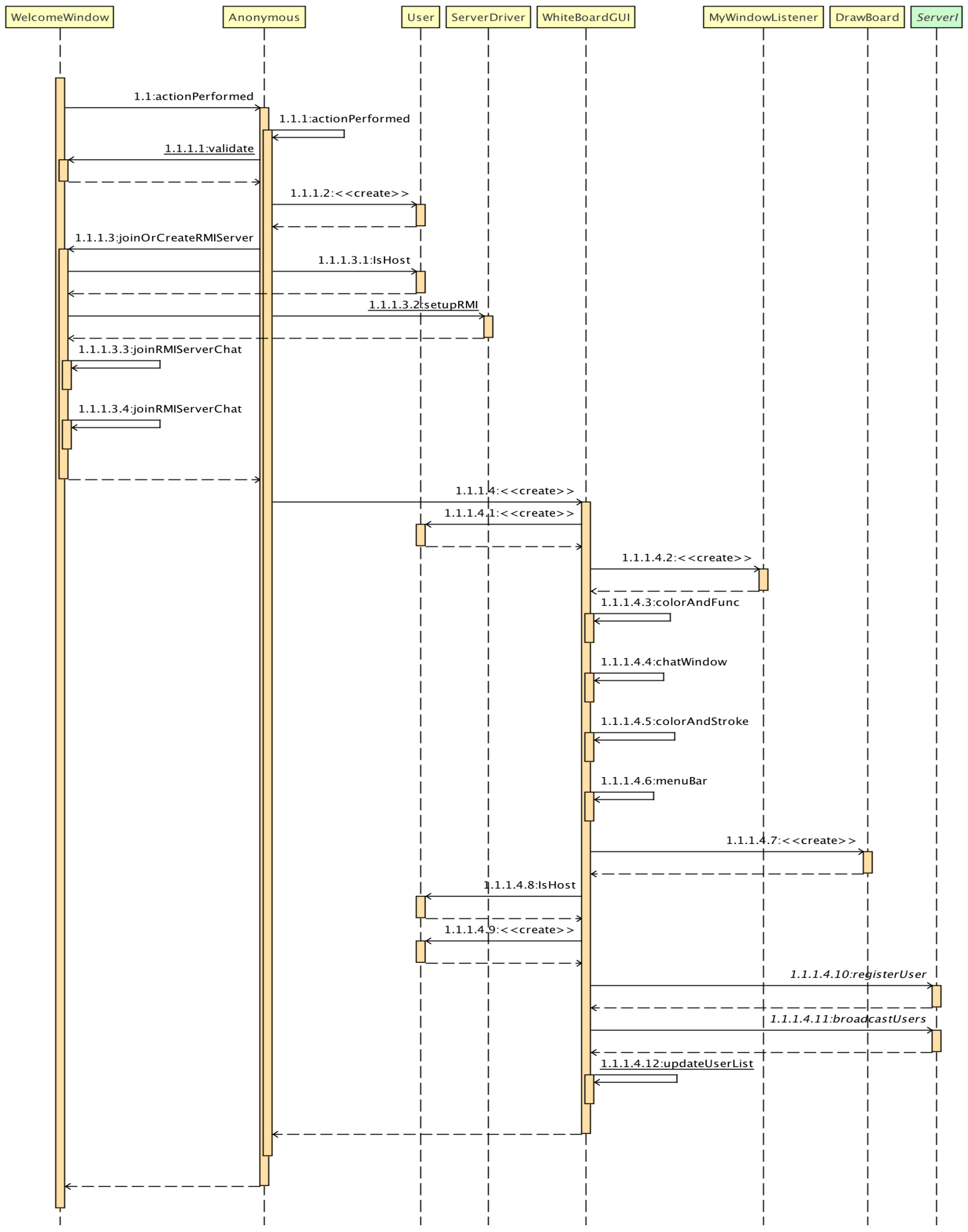
The methods contained in the interface and the base class, which implements the interface, are used to pass information through the server to all clients. This includes the chat messages and the drawboard, as well as management functionality such as kicking a client from the whiteboard collaboration.

Contributions

We, the project group, have all worked together on creating the distributed whiteboard. However, in order to avoid doing the same work twice, every group member had an area that they were primarily responsible for. Kasper worked on the RMI for the drawings, and did general code review, bug fixing during the project. Marc worked on the establishing the RMI connections, using the RMI for creating the Chat component and User Management Functionality. Navnita worked on the GUI, User Management Functionality and the Report. Xin created the GUI and worked on the User Management Functionality.

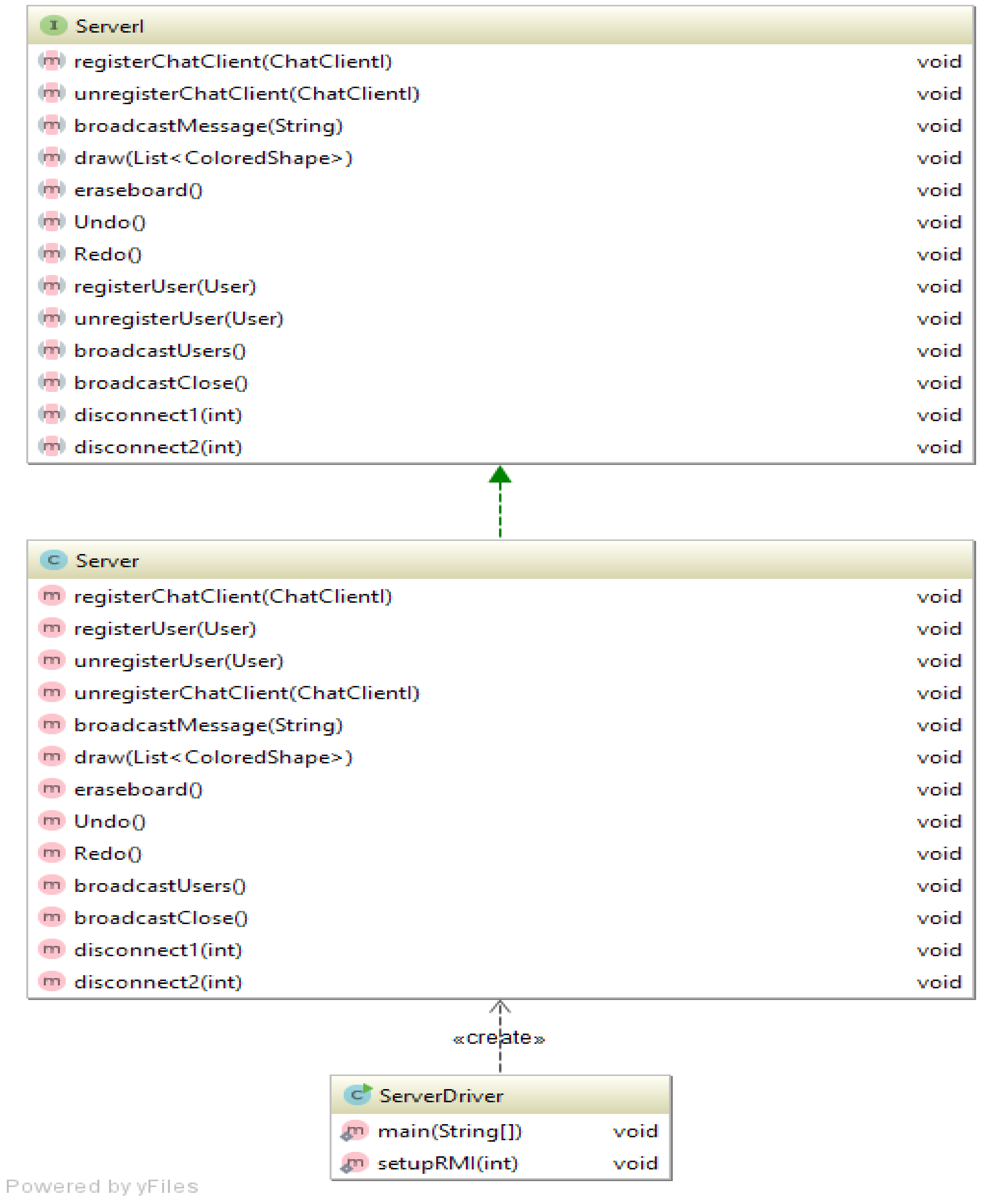
Appendix

VI.I Sequence Diagram

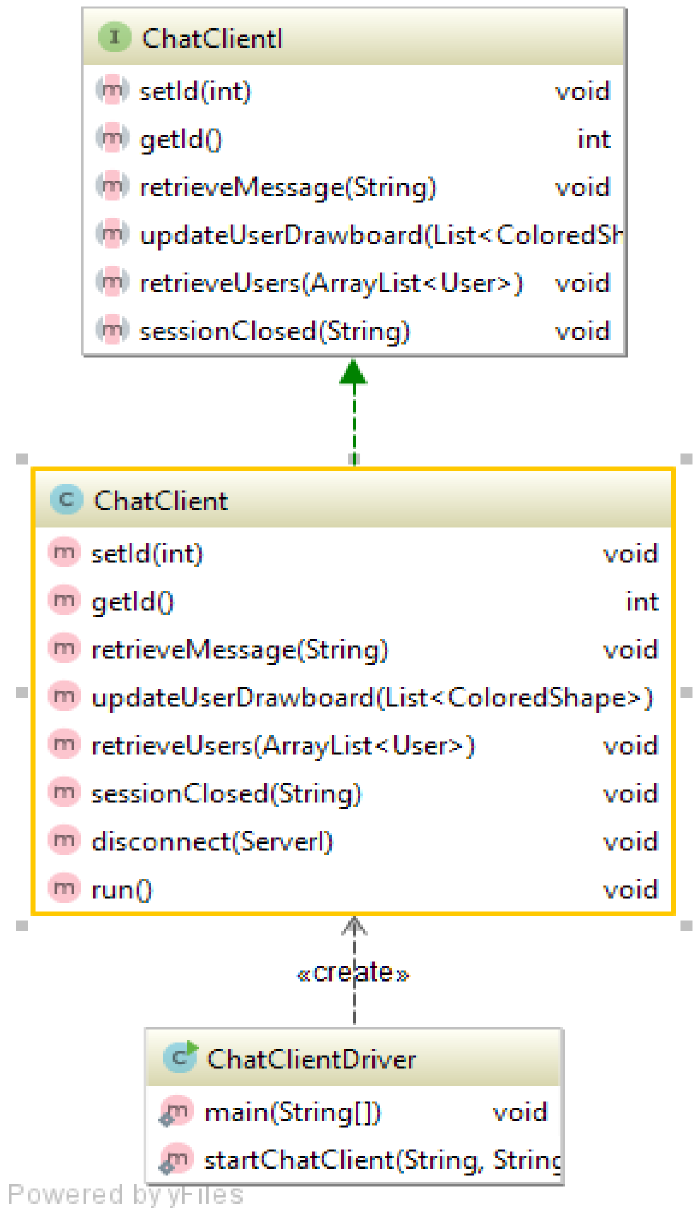


VI.II Class Diagrams

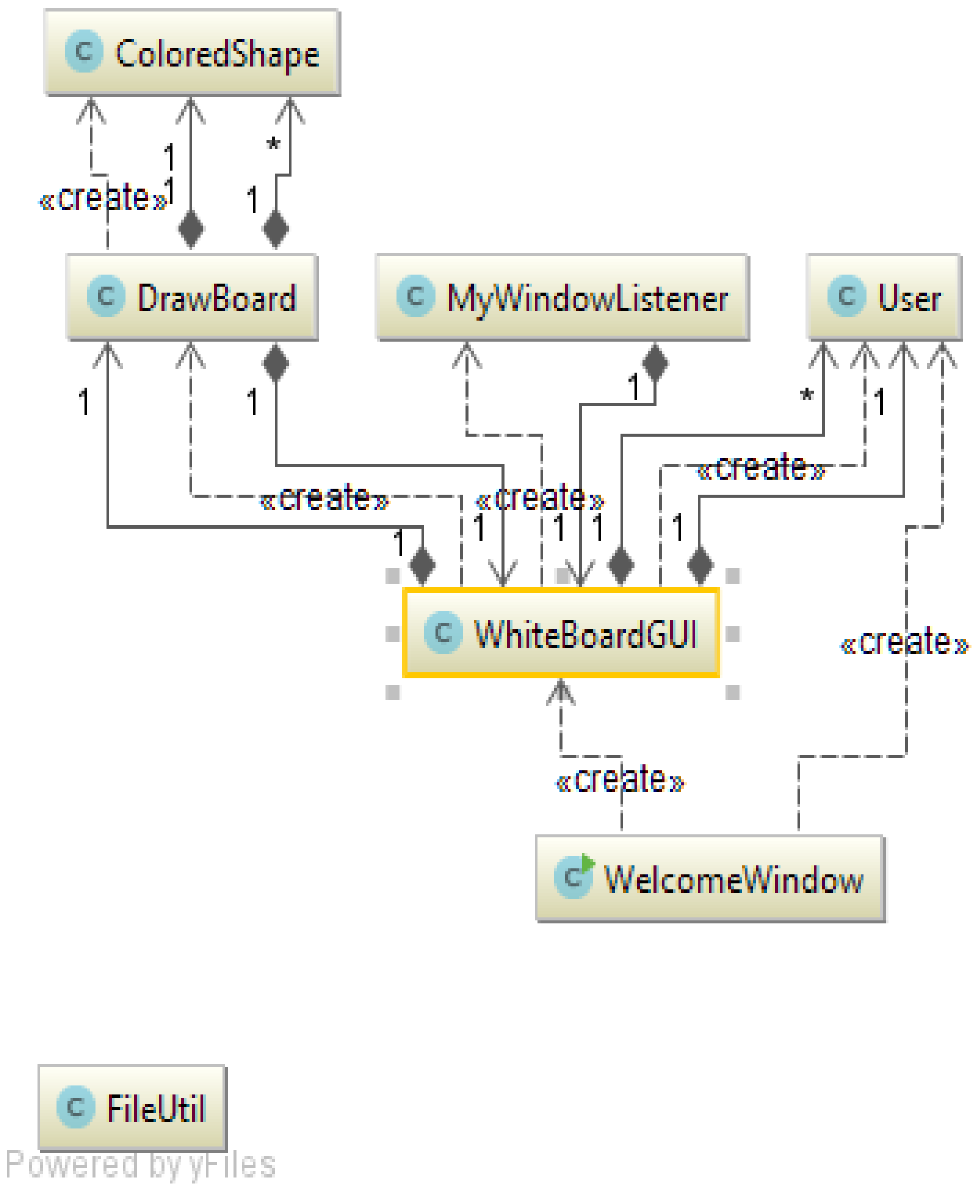
Chat Server



Chat Client


ChatClientI
setId(int) void
getId() int
retrieveMessage(String) void
updateUserDrawboard(List<ColoredSh
retrieveUsers(ArrayList<User>) void
sessionClosed(String) void
ChatClient
setId(int) void
getId() int
retrieveMessage(String) void
updateUserDrawboard(List<ColoredShape>)
retrieveUsers(ArrayList<User>) void
sessionClosed(String) void
disconnect(ServerI) void
run() void
«create»
ChatClientDriver
main(String[]) void
startChatClient(String, String
Powered by yFiles

WhiteBoard


ColoredShape
«create»
1
1
*
1
DrawBoard
MyWindowListener
User
1
1
1
*
1
«create»
«create»
«create»
1
1
1
1
1
1
WhiteBoardGUI
«create»
«create»
WelcomeWindow
FileUtil
Powered by yFiles