COMP90015 Distributed Systems Assignment 2

Name: Hongzhi Fu Student ID: 1058170

1. Introduction

1.1 Socket

In a network system, there are multiple servers and clients connecting with each other via some protocol, which involves the most two common protocols in transport layer, TCP and UDP. TCP, also called transport control protocol, is a connection-oriented protocol, provides reliable data flow, and have mechanisms to handle out-of-order and replicated data segmentation. Because of reliability of TCP, it can be applied to transfer a secret file, remote terminal access, etc. UDP is connectionless protocol that only transports several independent datagrams, which is very fast, but it does not guarantee the arrival and sequence of those packets. UDP also has many applications, such as video streaming, DNS service, etc. However, network protocols typically have a very complex structure. To address it, a network socket provides an interface that two entities can communicate with each other. With the encapsulation of object-oriented programming, we can create a simple, easy-to-manage and efficient network application. In TCP, we can communicate with a server via binding server's host name and port number that specifies which process we need to access.

1.2 Multithreading

In the real world, one server serving only one client machine is not applicable and sustainable. Instead, multiple clients will connect, send requests to a server concurrently, which requires server running multiple programs concurrently along with consistency. Multithreading is one technology that supports multiple clients accessing shared resources simultaneously. The model of multithreading server has two ports: one is for listening the upcoming client that requests for connection, another is for data transfer or providing other services. Initially, the server creates a socket which binds its hostname and port number, then waits for connection from clients. For each time client sends a connection request, the server will create a thread with a unique port number for further communication while keeping the original port number alive for accepting other client's connection request. With multithreading, clients are able to acquire, add or update shared resources concurrently. While in some situation, in particular, bank account information retrieval, concurrent operation will cause a big problem, so we need to introduce synchronization to handle each concurrent operation.

1.3 Shared Whiteboard

In this task, we will implement a shared whiteboard that held by a manager, who has privilege to create a new board, save the current canvas, open the previous canvas, handle incoming join request, so that only the authorized user can join the shared whiteboard, kick a specified user, and terminate the application. The whiteboard can draw whatever the user what to draw, such as typical shapes (including line, rectangle, circle, oval, etc), brusher, textbox, and eraser for removal. The style of drawing is also free-styled, thus user can choose the favorite color and adjust the stroke width of the shape. The last functionality is user can send a message to all other others by typing in the chatbox.

2. System Architecture

2.1 Whiteboard GUI

The whiteboard GUI has many components for displaying and several events to implement drawing shapes or paint on the canvas in different colors, which is shown in Figure 1.

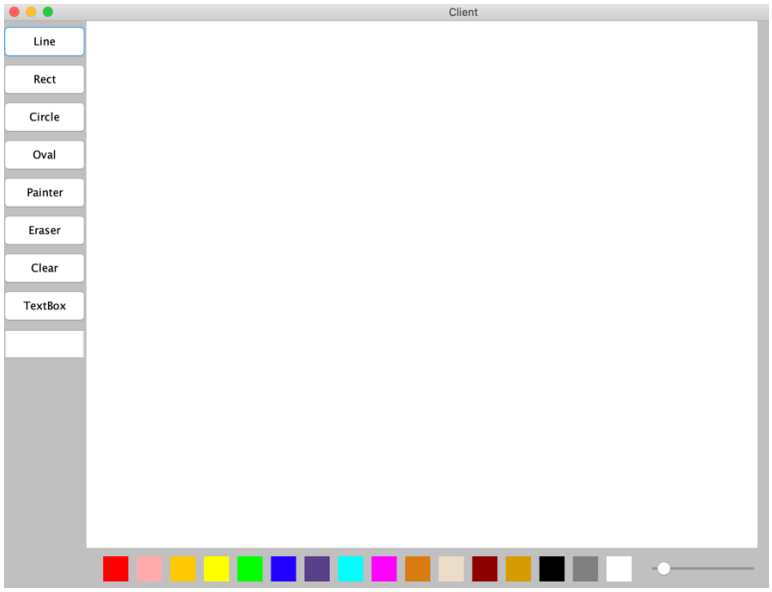

Figure 1. Whiteboard User Interface

The first implementation is the design of the user interface. As we can see from the above figure, we placed each drawing option on the left side and drawing style (color and stroke width) on the bottom, which contains 16 different colors and a slide bar that can control the width of the stroke in a shape.

The second implementation is to create listener events on all buttons, slider and the whiteboard. For each button, what we need to do is to implement the `ActionListener` interface, in particular, change the mode of which shape and which color we would like to use. The slider bar at the bottom has the same procedure as the buttons, except we must implement `ChangeListener` interface in order to get the value when dragging the bar back and forth. To implement events in the whiteboard, we should implement and override all methods relating to mouse events including `MouseListener` and `MouseMotionListener`. The logic behind is when one specific event is triggered, we obtain the coordinate of both x-axis and y-axis. For example, when we want to draw a straight line, we must have the coordinates of two end points, so the first coordinate x_1 and y_1 are returned when user pressed the mouse, which triggers `MousePressed` event, in the whiteboard and dragged the cursor to the other coordinate. When user releases the mouse, which triggers `MouseReleased` event, it returns the second coordinate x_2 and y_2. Therefore, we can invoke `Graphics.drawLine()` method to draw a line between any two points.

After implementing the two components, the client user can freely draw any shapes and texts they want. The effect of drawing is shown in Figure 2.

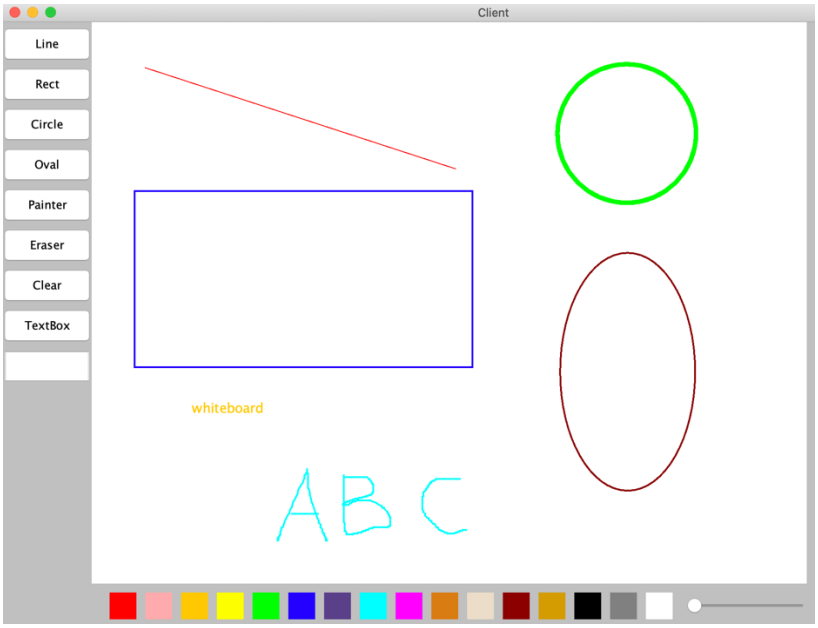

Figure 2. Drawing Effect of the Whiteboard

2.2 Functionality

The communication protocol we used for transferring texts and objects is TCP socket. In the server side, it creates a multithreaded server socket that waits for client connection. When a client sends a connection request message, the server will accept the connection and creates a new thread to handle the connection, such as data transfer, text communication, etc.

2.2.1 Join and Leave

In our case, the client cannot join the shared whiteboard before being approved by the manager. So when a user clicks Join button, the pop-up dialog box will inform the user to wait for the approval of the manager. When the manager receives the join request message from the client, the manager will decide whether to invite the user into the shared whiteboard. If approved, a name will be added into the user list of the manager, and reply the user that you are approved by the manager. Otherwise, the user will receive the rejection message and close the window automatically. The flow chart of a client joins in the shared whiteboard is shown in Figure 3.

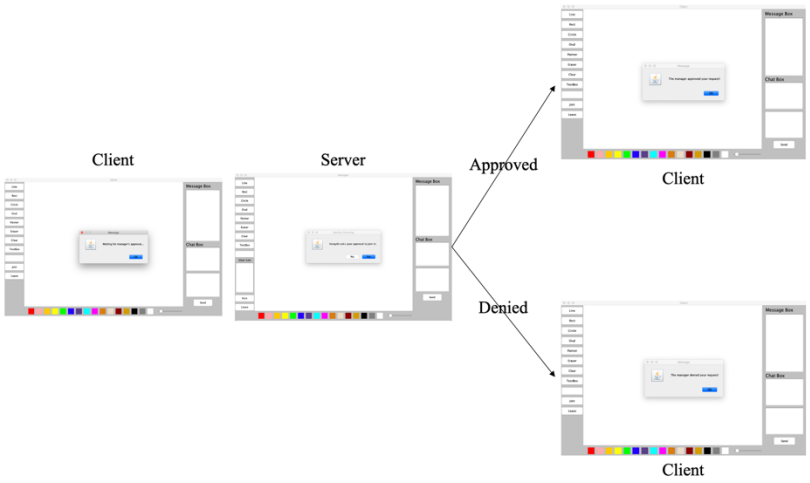

Figure 3. Flow Chart of a Client Joining the Shared Whiteboard

Each client has a unique identifier that the manager and other clients can identify them. However, if one user uses an identifier that duplicates with the current identifiers in the user list, it will inform

the user to replace another name by typing or generating a random name from candidate name list. The whole process is shown in Figure 4.

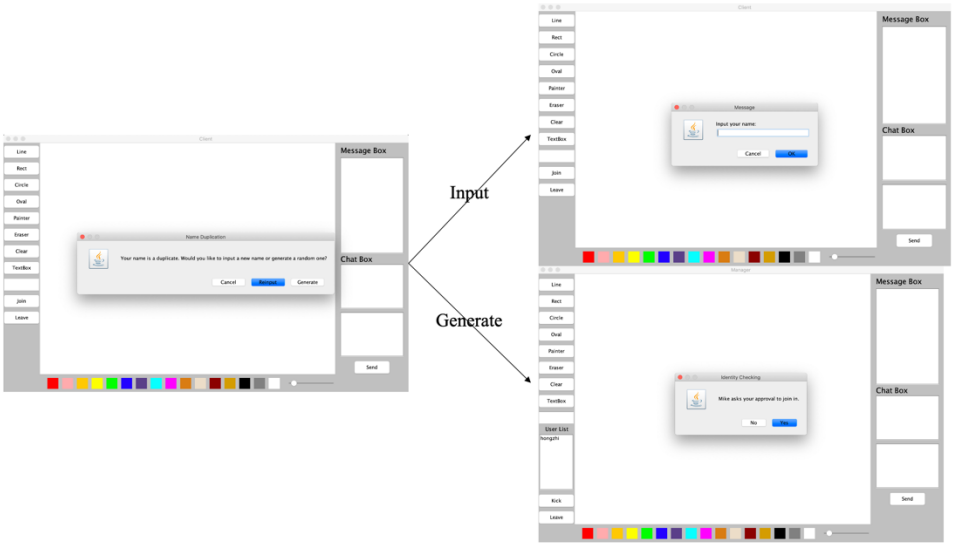

Figure 4. Flow Chart of a Client Handling Duplicate Name

Clients can leave the shared whiteboard whenever they want by clicking Leave button. After clicking it, a message box will appear in the client UI that ensures whether they really want to exit the current whiteboard. If the user clicks yes, the client UI will close the window automatically, while the manager UI will update the current user list. The process of a client leaving is shown in Figure 5.

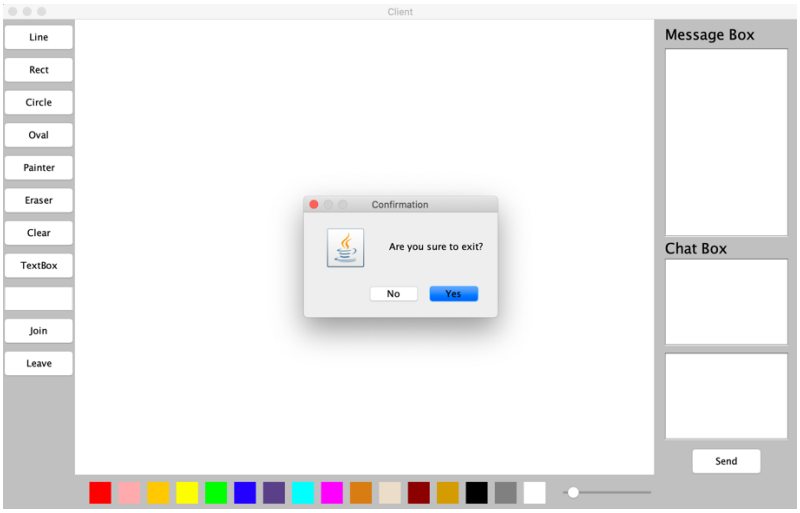

Figure 5. Dialogue Message When User Clicks Leave Button

2.2.2 Message Broadcast

In order to achieve chatting and noticing what other users are drawing something, we need to broadcast the message so that all other users will receive the same message. The broadcast mechanism is handled by the server, where whenever a join request is approved by the manager, the server will add the key-value pair, which is username-output stream, into a hashmap called `all_writers`. Thus, the broadcast can be done by the server by iterating each writer stored in `all_writers` and send a message to all users. The illustration of message broadcast on chatbox and message box is shown in Figure 6.

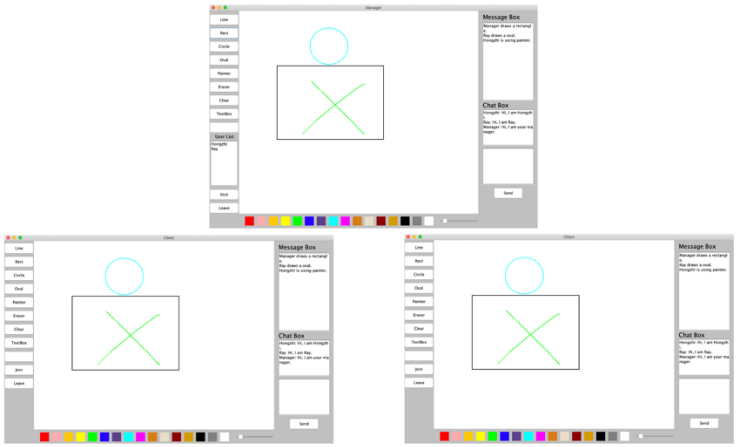

Figure 6. Illustration of Message Broadcast

2.2.3 Image Sharing

To achieve a distributed whiteboard, we must share the content of the whiteboard as an image file and share the image formatted as `.png` to other users. As the rule suggests, only the manager can create a whiteboard, unauthorized users have no privilege to draw something on the whiteboard until they receive a message from the manager. Our architecture of the shared image is implemented by the distributed file that handled by the manager. When a manager creates a new whiteboard, it will create a temporary file in a specific folder, and the file name is composed of the name of manager and the creation time in the `./temp` folder. For example, if the manager is called Bob, and the whiteboard was created at 15th of May, 2021, the image name would be `Bob: 2021-05-19.png`. Then, when a new joined user is approved by the manager, the manager will send the path of the shared image, so the client can read the shared file for distributed purpose.

2.2.4 Design Diagram

The UML class diagram is shown in Figure 7.

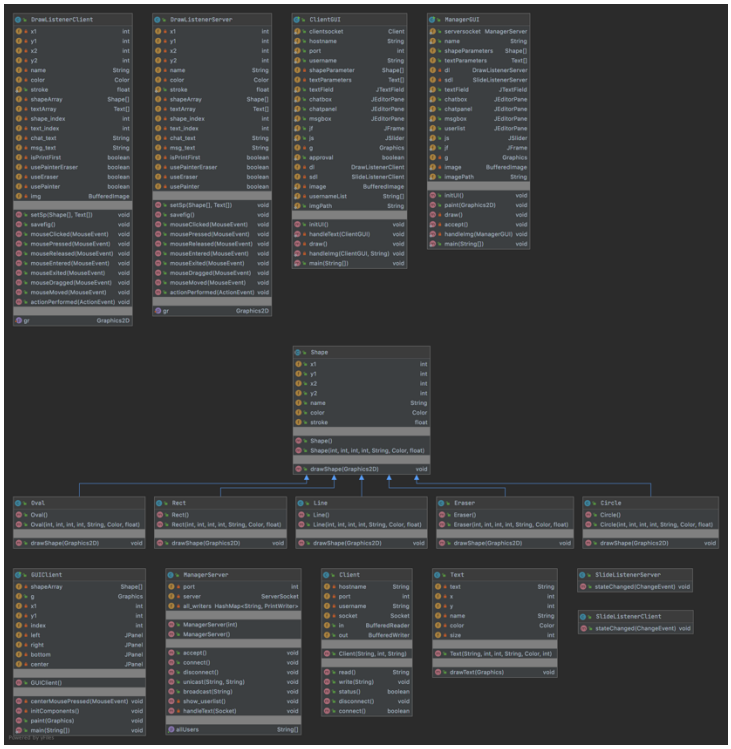

Figure 7. UML Diagram for All Classes and Interactions

3. Advanced Features

3.1 Chat Window

The chat window is based on the message broadcast shown in section 2.2.2. Whenever the client establishes a connection with server, the server will add the client's output stream into a hashmap, so that when invoking broadcast method, the server can iterate the hashmap to get all writers corresponding to each user and send the messages. The problem is how does the server distinguish messages between normal contact message and message for request whiteboard sharing. The solution is using predefined header frame, which is the 17 random combinations of characters and numbers. The following code snippet shows the difference of header frame message for different purpose.

```
if (msg != null){
    // chat window message
    if (msg.startsWith("m3oimru3289mr2384")){
        String processed_msg = msg.substring(17);
        ManagerGUI.chatpanel.setText(ManagerGUI.chatpanel.getText() + processed_msg + "\n");
        broadcast(msg);
    }
    // message box message
    else if (msg.startsWith("c394c3j2cm2390sf9")){
        String processed_msg = msg.substring(17);
        ManagerGUI.msgbox.setText(ManagerGUI.msgbox.getText() + processed_msg + "\n");
        broadcast(msg);
    }
    // client joining message
    else if (msg.startsWith("jx82cr89sfd4jc392")){
        String identity = msg.substring(17);
        if (all_writers.containsKey(identity)){
            int result = 2;
            out.println("jx82cr89sfd4jc392" + result);
            continue;
        }
       int result = JOptionPane.showConfirmDialog(ManagerGUI.jf,  message: identity + " asks your approv

       if (result == 0){
           all_writers.put(identity, out);
           show_userlist();
           PrintWriter writer = all_writers.get(identity);
           writer.println("jx82cr89sfd4jc392" + result + ManagerGUI.imagePath);
       }
       else if (result == 1){
           out.println("jx82cr89sfd4jc392" + result);
       }
    }
    // client leaving message
    else if (msg.startsWith("v23t89rvu348908so")){
        String identity = msg.substring(17);
        all_writers.remove(identity);
        show_userlist();
        return;
    }
}
```

Figure 8. Code Snippet of Server Handling Different Types of Messages

3.2 Message Window

In addition to chat window for users communicating with each other, this program also supports message window to show which one does what. For example, if a user draws a line in the shared whiteboard, the message window box will show that someone draws a line and broadcast to all users. Thus, for each mouse event, the message will be broadcast to other users, so that the server will also need to specify what types of message it belongs to by identify the header frame.

3.3 Kick Users

Since the manager can handle which user has privilege to join the whiteboard, it should also have a function to kick a user. When the manager clicks the Kick button, it will pop up a dialogue message that shows a list of all users. One can select a user to kick off, and the server will appear a confirmation message to ensure the selected user will be kicked off. Another choice is to kick all users by broadcasting a special message header. The flow chart of kicking users is demonstrated in Figure 9.

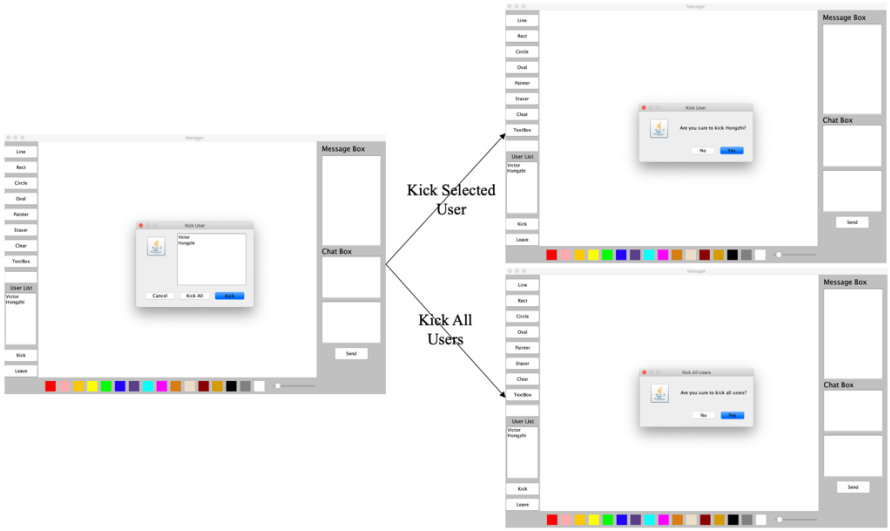

Figure 9. Flow Chart of a Manager Kicking Users

3.4 File Menu

Another advanced feature is manageable file menu that controlled by manager. By adding menu bar on the frame, we can implement the graphical interface shown in Figure 10.

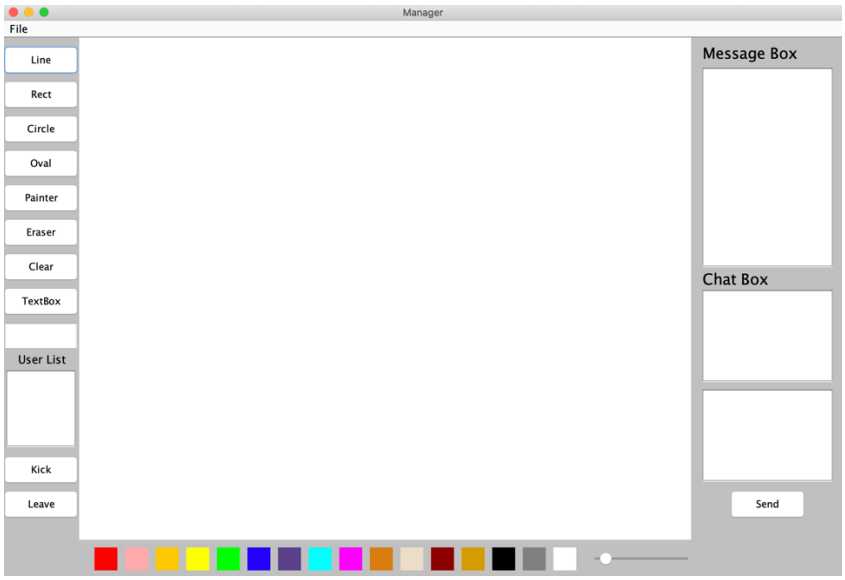

Figure 10. Manager GUI with File Menu

3.4.1 Create a New Whiteboard

The first function is to create a new whiteboard, so that there are multiple managers with multiple users. The implementation of new whiteboard creation is to execute java command directly in the "New" ActionPerformed event. To avoid two managers sharing the same port number, we support port number and manager name input, before actually creating a new window. The process of whiteboard creation is shown in Figure 11.

Figure 11. Process of Creating a New Whiteboard

3.4.2 Open an Existing Image

When clicking the open button in the file menu, the program will guide the manager to select a file from the current directory in the file dialogue. To make the program robust, we use file filter in which only the correct types of file (e.g. PNG, JPG, JPEG, etc.) can be selected. The process of opening an image file is shown in Figure 12.

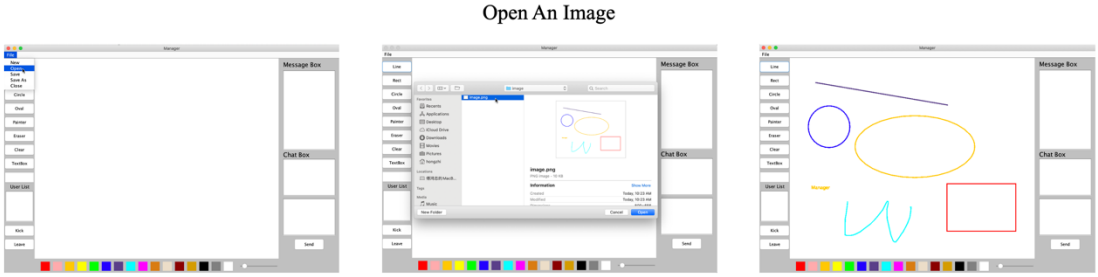

Figure 12. Process of Opening an Existing Image

3.4.3 Save and Save As

Manager can also save the existing file and specify the directory in his/her computer. If the manager has not saved the image before, the save dialogue will pop up and when the manage save the image successfully, it will have a message dialogue showing the image has been saved. If the manager has saved before, it will only show a dialogue that the image has been saved. The process of saving image is shown in Figure 13.

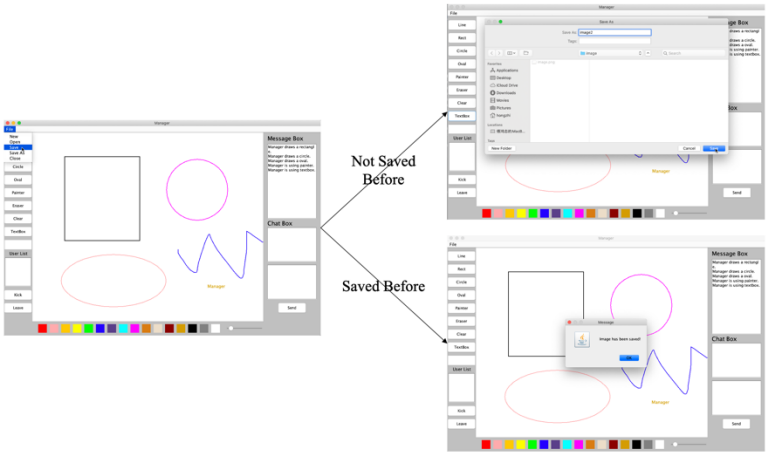

Figure 13. Process of Saving an Image File