COMP90015: Distributed Systems
Assignment 2: Distributed Whiteboard
2022, Sem 1
Mylan Li – 695788

System Architecture

This project uses a simple Client-Server model, with a single server. As this architecture was suitable for all the requirements of the project, more advanced models were unnecessary. Furthermore, due to the requirements of there needing to be a host/manager it made sense for there to be a single server, rather than a peer-to-peer model.

Communication Protocols (and their implementation details)

The project uses RMI to facilitate communication between the server and clients. Clients can only communicate to the server; all communication “between” clients is relayed through the server, with the server containing the logic to send communication onwards. The server also broadcasts its whiteboard functionality related communication to all connected clients, with specific client communication limited to the initial connection request and when it kicks a client.

RMI was chosen over sockets due to their more streamlined approach to communication between servers and clients, which of interest for concurrency of a shared whiteboard, rather than having all functionality needing to be passed via string-like message formats like JSON. This was especially relevant for sending images to the clients such as when opening a saved image.

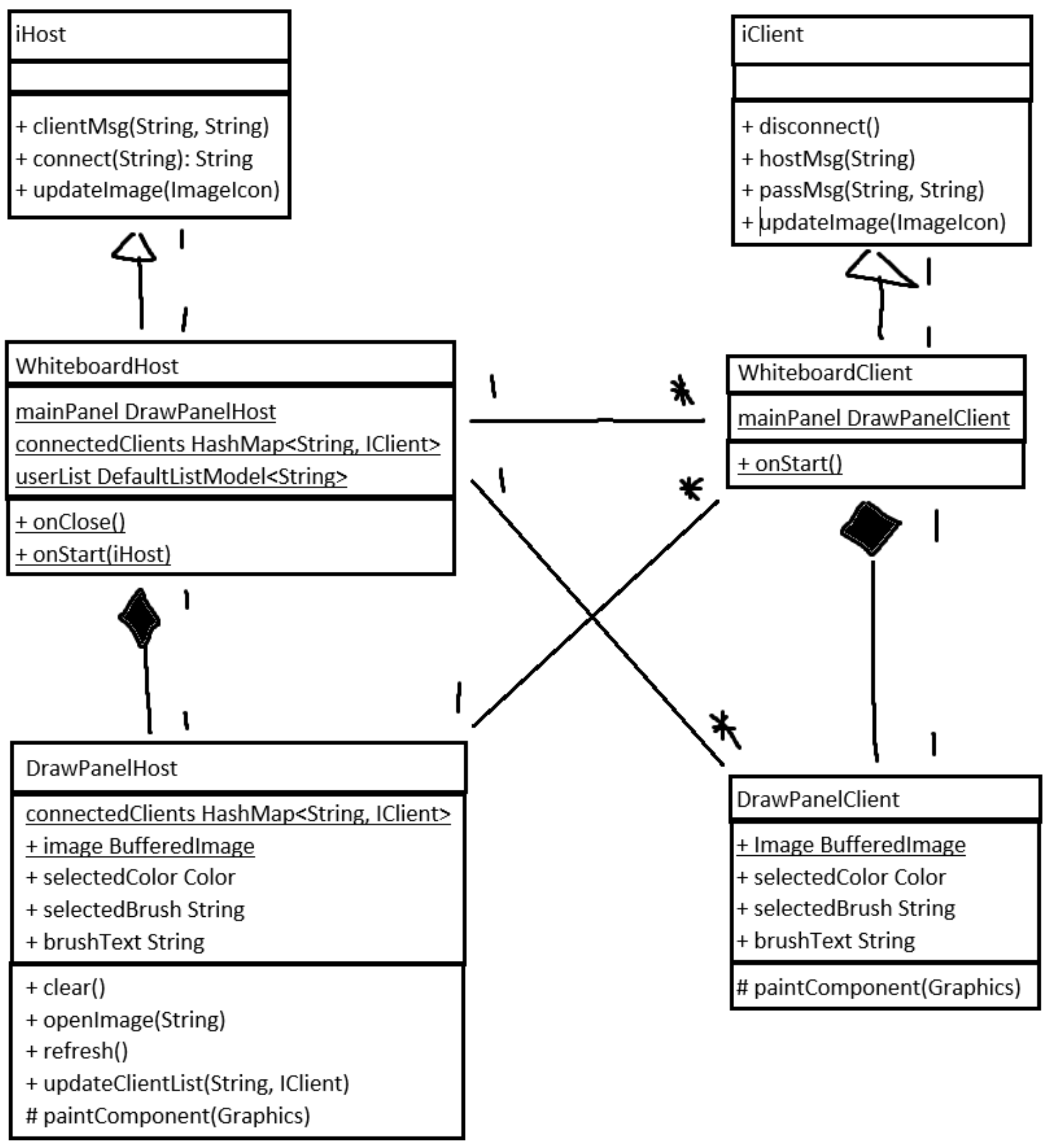
RMI also inherently supports concurrency between the users. Any drawings that clients do are automatically passed onto the server which in turn relays it to the rest of the clients, as when as broadcasting any changes that the host user does. Java inherently seems to have multi-threaded behaviour for the client main thread and the RMI methods, though this is discussed more in the analysis and innovation section.

The project also relies on separate classes for the whiteboard drawing functionality for both server and clients. This is not ideal, as it does lead to suboptimal behaviour, with both the main server class and its drawing helper class maintaining technically separate records of connected clients. However, it allowed for the project to be completed on time and meets the specified requirements, so further refinement is discussed in the analysis and innovation section

Specific details about implementation of communication methods will be discussed in the sections below.

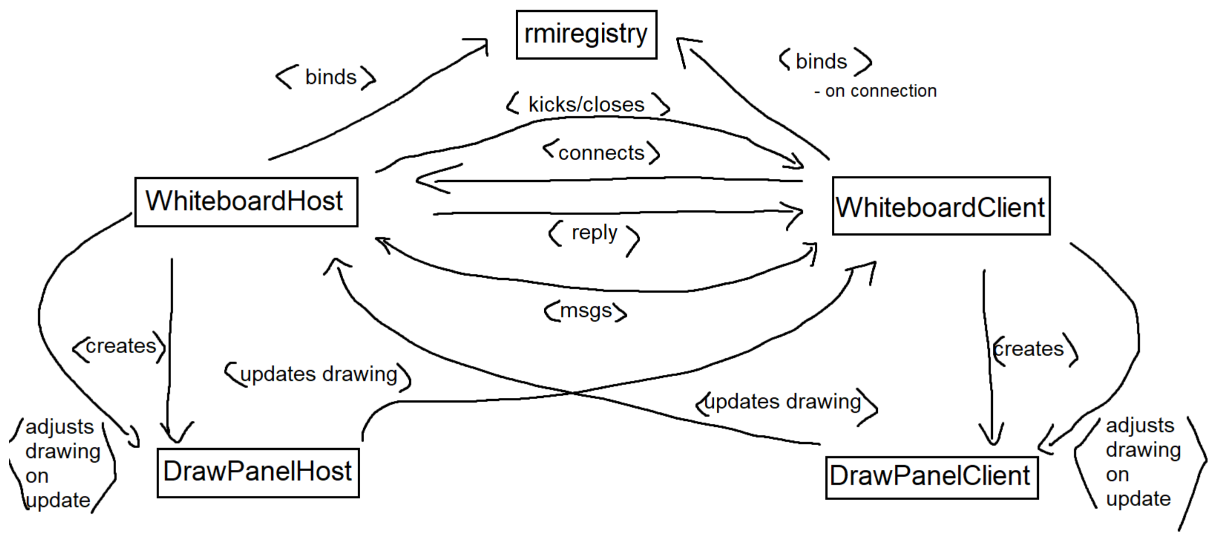
Class design and Interaction Diagram

The diagram below shows a simplified, high-level overview of the classes in the project, detailing the important methods and attributes involved in communication.



Of note is the communication between the DrawPanel classes and the respective server/clients, as mentioned in the previous section and to be expanded on later.

Diagram of simplified interaction between host and clients



Analysis, Innovation and Future Improvements

DrawPanels

During development, drawPanel was developed as part of the initial phase. When it came to implementing it with the skeleton on which the client-server interactions had been developed, after multiple failed attempts, this was the successful integration. The respective drawPanels thus are handed information with which to update the canvas and communicate changes to the server/client, with the main class only directly communicating about user management and chat.

This is suboptimal, with a messy design that causes some problems (of which one of the more visible ones will be addressed later). While it does get the job done and works smoothly for the most part in an ideal context, future improvement would see the project redesigned/redeveloped such that there isn't a need for the drawPanel to be so separated from the main server/client class.

User disconnections

When a client disconnects, there is no immediate feedback to the server that this has occurred. This is reflected in the connected users list, which does not change when a user disconnects. The host program does have logic to account for this, updating its map of connected users when a communication with the specific client fails. Ideally though, a future improvement would be to fix this lack of concurrency.

There is also an issue due to the drawPanels being separate related to this. As the drawPanelHost has its own list of connected clients, it fixes that on its own, but does not have a way to communicate this to the main host program, requiring it to discover the disconnection on its own. Again, not ideal and would a future improvement to be made.

*I realised that that through my implementation of having the host program notify and close the client programs, that there I could include logic for the clients to notify the host when they disconnect at user action. Could be used as part of future improvements.

Approving Users

A known bug is that due to the way that users request permission to join the shared whiteboard through dialog boxes, it can result in the situation where two clients join with the same name.

This is the bug that is closest to a main feature, and should be addressed in future improvements.

Notification to Users of disconnection

When users are disconnected by the host, the notification for the host that this has occurred doesn't display, which seems to be related to unbinding the client message function that it uses as part of the process. Could be improved on.

Closing host

When the host is closed, currently is notifies and closes the clients one by one, as part of the design choice to prevent the clients from crashing on their own when attempting to access the shared whiteboard that is no longer there. One quirk is that as part of a small delay so the user can see the notification that they are being disconnected, the main host also has to wait this time out before the method finishes. This adds up to a noticeable delay, so could be improved on.