2019

Distributed Shared Whiteboard

AAYUSH MEHTA (1105081)
ABHIJEET SINGH (1094984)
ANOUSHKA K. DOCTOR (1012827)
SANJAY GOVINDRAO KULKARNI (1089503)

1. Introduction

Coulouris et al.[1] defines a distributed system as one in which software or hardware components located at networked computers communicate and coordinate their actions only by passing messages while maintaining concurrency, without a global clock and the ability to handle independent failures. This report aims to highlight a simple implementation of a distributed system – a shared whiteboard.

The problem defines a client that should be provided with options to draw a multitude of shapes on the whiteboard such as a circle, oval, rectangle or be able to draw "freehand", input text or erase his drawing from the canvas provided by the whiteboard. The first person to join the session is awarded the position of the manager. The manager decides which client can join the current session and also has the ability to kick a client off the session. He alone has the privilege of opening a new canvas or saving a canvas. The whiteboard is shared, i.e. all clients see the same canvas and draw on the same canvas. All changes must be reflected on the canvases used by all clients with minimal delay. A chat functionality must also be implemented giving all users the ability to communicate with each other. A simple Graphical User Interface (GUI) must be created to give the user the ability to use all functionalities. Java is to be used to implement the described system.

This report delineates the architecture of the defined system, protocols for client-server communication, the components and a critical analysis of the system designed to address the defined problem. It also covers the excellence and creativity elements achieved by the system.

2. System Architecture

2.1. Technology used

Overall, the problem is to create a distributed system. Many technologies can be employed to build such a system. However, for this implementation, Sockets have been used to construct the distributed system. Socket programming is well defined in Java, easy to implement and very efficient. The "manager" is essentially the server and is implemented by using the ServerSocket class which requires a port number for the manager to setup on. The Socket class acts as the client and requires the same port number that the server started on along with an IP address to setup.

Sockets were chosen for this particular problem over other technologies such as Remote Method Invocation (RMI) as they are the offer more control. Using sockets, one can control in what form data is exchanged between the server and the client, hence offering more flexibility. Also, it is easier to publish objects using Sockets rather than using RMI and has very less overhead. All these advantages make sockets the perfect choice for the application to be built.

2.2. Threads for chat and draw functionalities

In the aforementioned problem, the client must be able to draw on a canvas as well as chat with other clients on the network. Two different sockets were used to setup each of these functionalities. This means that when the client wishes to "chat", a new thread is spawned on a different socket using the thread-per-connection architecture. The thread-per-connection architecture spawns a new thread each time a new user connects

to the network and is destroyed when the connection is terminated. The same happens when the client first starts to draw on the canvas. Both these functionalities hence utilise two distinct sockets. This design decision has been made such that when a user uses each of these features, they do not hinder the operation of one another and offer a seamless user experience. Operating on two different sockets makes sure that both features are working with optimal capability and the failure of one thread doesn't necessarily stop the user from using the other feature.

2.3. Communication protocol

Two of the most popular protocols for client-server communication are Transmission Control Protocol (TCP) and User Datagram Protocol (UDP). Out of the two, TCP is connection-oriented and provides reliable communication between the client and server by making sure that each end receives the intended segments and provides recovery of segments which are damaged, duplicated or lost. In order to fulfil the requirements of the system, TCP was chosen as the communication protocol to provide reliable communication. TCP makes sure that the message is sent successfully in a time-efficient manner.

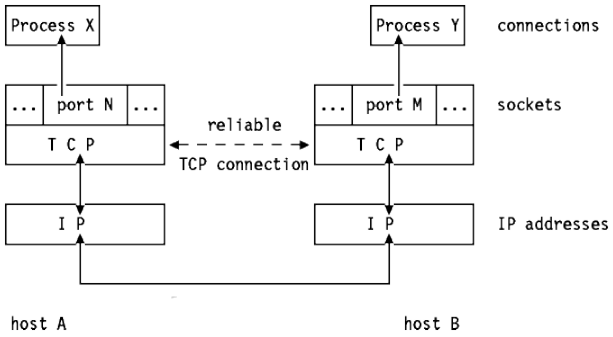

Figure 1. TCP connection (Sinha, 1998)

3. Components of the system

The class diagram delineates all classes required to deploy a solution for the defined problem. Appropriate error handling mechanisms are put in place to give the user a clear indication of the errors generated. A few key classes and their operations are elucidated below:

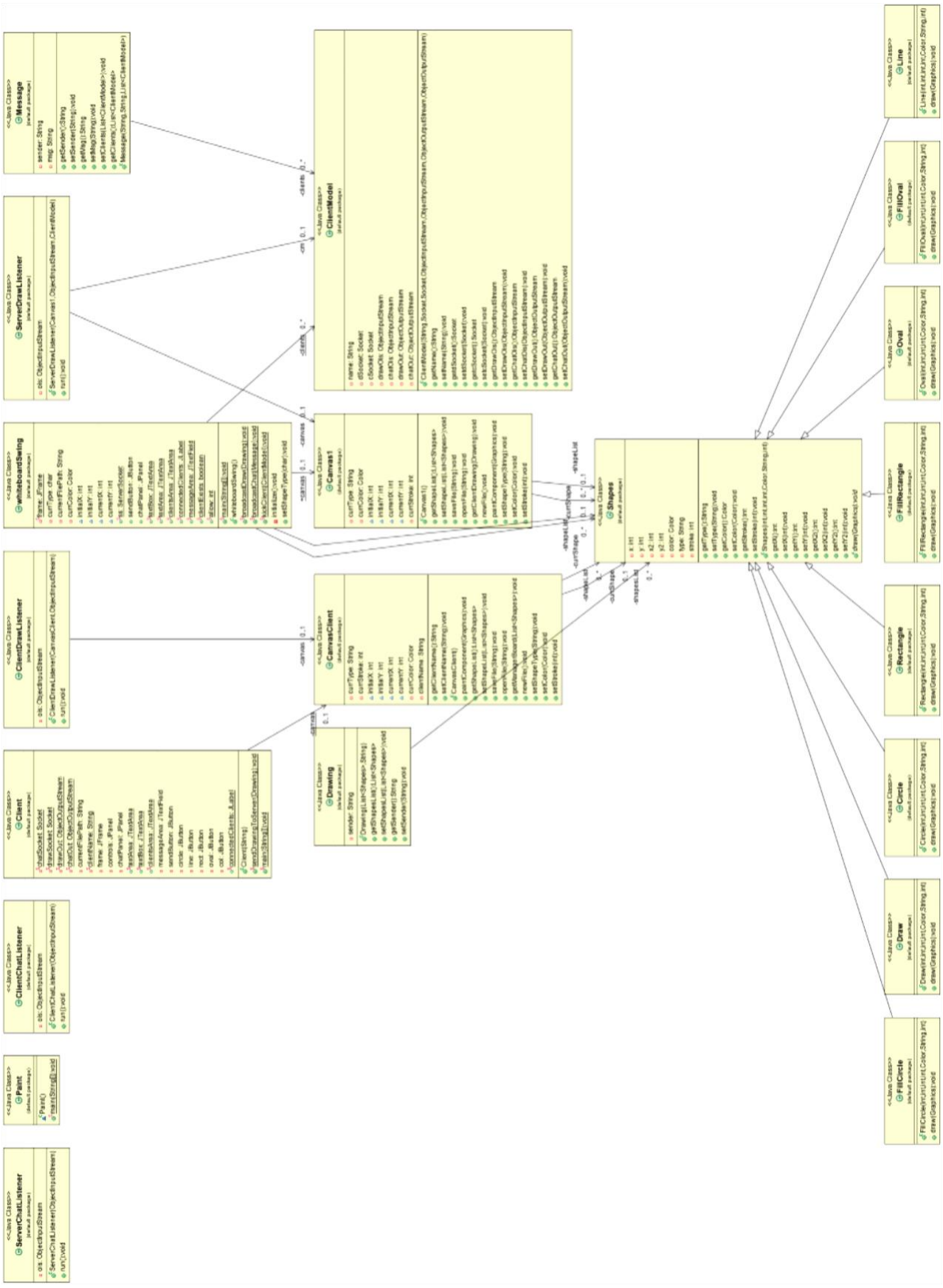

Figure 2. Class Diagram

3.1. Shapes Class

This is an abstract class which contains methods and variables required to draw a given shape. The variables are the properties of each shape available to the user. The methods are setters and getters of each variable along with a draw method. All shapes that are available to the user (Rectangle, Circle, Oval, Freehand drawing, Eraser) inherit this abstract class and hence override the draw method. Each shape is its own different class and defines its own implementation of the draw method.

3.2. Canvas and CanvasClient Class

The Canvas class essentially provides the "drawing on the whiteboard" functionality to the "manager" (server) whereas the CanvasClient class does the same for the client. Two different classes where created as functionalities such as the ability of saving a drawing or opening a new drawing are only available to the manager. These classes contain the implementation of the ability to draw on the shared whiteboard.

3.3. Client/Server Chat/DrawListener Class

These classes are in-charge of capturing the chat/drawing sent by the manager or the client. The ClientChatListener captures the chat message sent by the manager and broadcasts it to all the other clients. Similarly, the ServerChatListener captures the chat message sent by any client. The ClientDrawListener captures any shapes drawn by the manager and broadcasts it to the other clients while the ServerDrawListener captures any shapes drawn by any client.

3.4. Client Class

This class allows users to join a session hosted by the manager. The IP of the server acts as an input and hence connects a client to the session. In order to join a session, the user is also prompted to enter a unique username. If the username is valid, the user is allowed to join the session. If the username already exists, then the user is prompted to enter a new username.

There are two distinct high-level features offered to the user – draw on the whiteboard and chat with other peers in the session. For each of these functionalities a new thread is spawned, each on a different socket. This gives the user a seamless experience.

3.5. WhiteboardSwing Class

This class gives the implementation of the "manager" (server) of the system. Each time a client wishes to connect, the manager has the option to accept or reject the client request. The manager also has the extra capabilities of opening a new drawing, saving a new drawing and kicking an already connected client out of the session. A single instance of the manager is created, and it handles both high level functionalities of drawing on the whiteboard and chatting with other peers. The server allocates two different sockets for the same client to work on, each socket dedicated to a specified feature.

3.6. Overview of the interaction between components

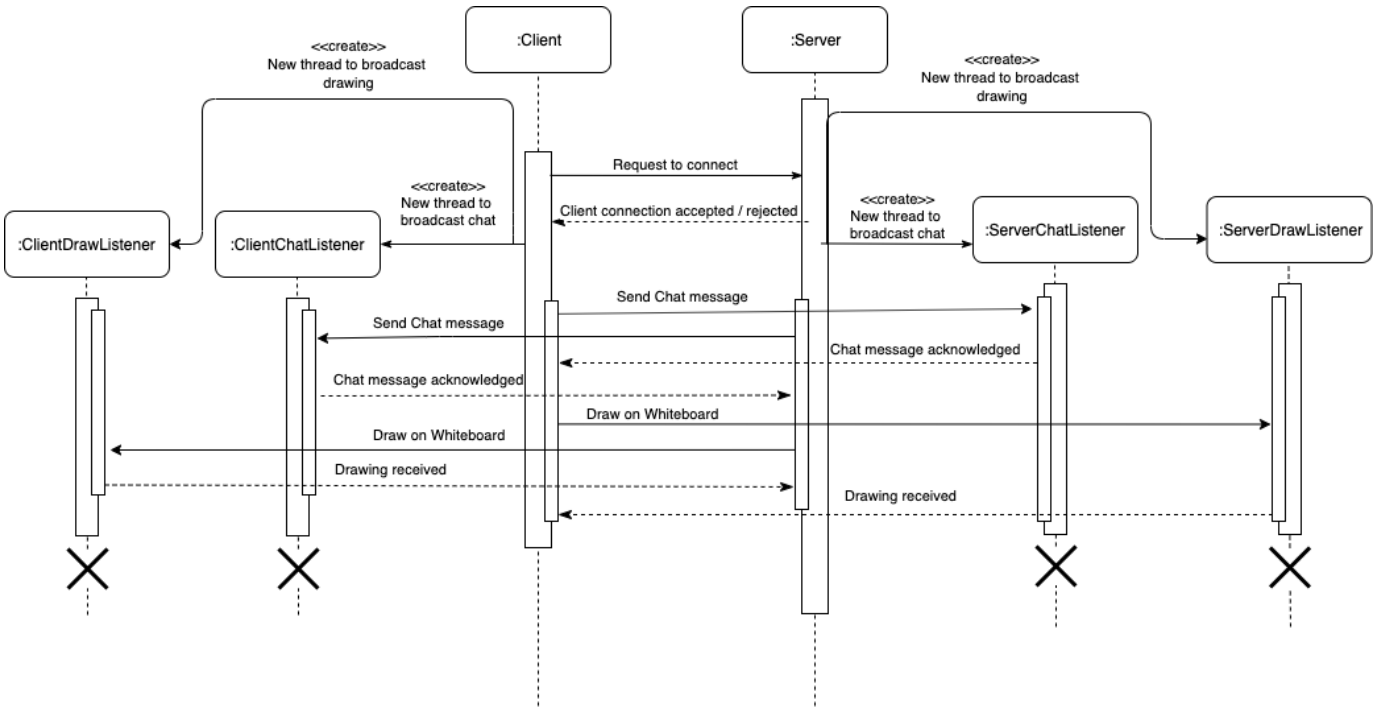

Figure 3. Sequence diagram

4. Critical Analysis

4.1. Selection of TCP for communication

TCP has been chosen as the communication protocol. This is a great choice in context of the problem. TCP provides reliable communication, and this satisfies the requirements of the system. Also, if the system were to be scaled, this protocol would prove to be extremely efficient as it has various error-control and congestion-control mechanisms in place.

4.2. Design of components

The system has been designed such that it is scalable and gives the user a simple GUI to utilise all the functionalities. We have implemented a design which utilises two distinct sockets to handle the two threads spawned for handling the drawing and the chat functionalities. This implementation makes the system quite scalable. Also, the technique used to broadcast the manager and the client boards with each other is such that there is no lag for either of the parties. The drawing appears to all the parties in real time and changes are reflected immediately.

Our design also stores the drawing as a list of objects. This gives our system the added security such that the drawing cannot be opened by any other application. In order to store the drawing as a list of objects, we have serialised the ArrayList and hence stored that list. This file is then de-serialised when the manager wishes to open the stored file. The file is unreadable by any other application than the proposed system.

The system also utilises ObjectInputStreams and ObjectOutputStreams over DataInputStreams and DataOutputStreams. As our system requires serialisation, the ObjectStream classes offer better performance and also provide with the added ability to not model the data to be stored separately.

The implementation of chat and drawing listeners also gives an added advantage to the system. Each of these listeners have a specialised capability and hence their functionalities do not get mixed. The chat listener is in-charge of capturing the chat messages sent by the server and the client while the draw listener is in-charge of capturing the shapes drawn by either. This provides a highly modular design for the system hence making it easy to spot bugs.

4.3. Design of GUI

The design of the GUI is simple and intuitive. It gives the user a clear idea about the features offered. The chat window also displays clearly the messages sent by each user accompanied by their identities. However, a more sophisticated GUI could have been employed to make the system more suited to the real-world.

5. Contribution

Name and ID	Contribution Area	Overall Contribution to the Project
Aayush Mehta - 1105081	Designed server-client architecture. Worked on synchronizing the whiteboard between the client and server. Contributed to the report.	25%
Abhijeet Singh - 1094984	Created free hand drawing and erasing functionalities. Added the chat functionality for the system. Contributed to the report.	25%
Anoushka Doctor - 1012827	Created functionality of drawing shapes on the whiteboard. Worked on file handling functionalities. Contributed to the report.	25%
Sanjay Kulkarni - 1089503	Created GUI for the system. Worked on the "kick-off" functionality and displaying peers to all clients. Contributed to the report.	25%

6. Excellence Elements

The design of the system has been carefully thought out and include the following excellence elements:

6.1. Error-handling

The errors are displayed to the user in a simple succinct way such that they can be easily understood, and the user can easily rectify the errors.

For example, when the manager abruptly quits the session, an error is shown to all clients "Server disconnected! Restart application".

6.2. Structure of report

The report is well formatted and is easy to read. All sections clearly indicate their purpose therefore, giving the reader a clear indication of the content of each section.

6.3. Necessary figures

The report consists of figures that help the reader better understand the concept and grasp the design of the system. The class diagram and sequence diagram give the reader an in-depth technical view of the system.

6.4. Critical Analysis

This section provides a clear explanation to whether the design choices made for the system are advantageous or disadvantageous. It also provides appropriate arguments that give the reader comprehensive knowledge about each choice made.

7. Creativity Elements

7.1. Mechanism of storing files

Our system stores the drawings created by the user as a list of objects instead of storing them as images (.png, .jpg, etc.). Each shape, change made on the canvas is recorded as a list of objects and appropriate additions and deletions are made based on actions performed by the client. This ensures that only our system is capable of opening the stored file hence adding a layer of security.

7.2. Concurrency between manager and clients

There is no lag between the manager and client boards. The entire system runs smoothly with all changes being broadcast in real-time. This is attributed to the unique design of our system. Our system only broadcasts the changes made to the whiteboard instead of broadcasting the entire drawing over and over again. This optimization gives the system a sophisticated and potentially deployable quality.

8. Conclusion

The problem defined was to implement a shared whiteboard application based on a distributed system. Designing this system gave a small taste of large-scale distributed systems, their architecture and their working. The system was designed such that the manager (server) has the ability to allow users to join the session while the users had the ability to draw on the whiteboard and chat with other peers. Sockets were used to bring the system to fruition along with TCP communication protocol. Threads were spawned for two of the functionalities offered to the user on two distinct sockets. The system in general is well designed and also has a good error-handling mechanism.

9. References

1. Coulouris, G., Dollimore, J., Kindberg, T., & Blair, G. (2012). *Distributed Systems - Concepts and Design* (5th ed., p. 294). Addison-Wesley.

2. Sinha, S. (1998). Two processes communicating via TCP sockets [Image]. Retrieved from http://www.ssfnet.org/Exchange/tcp/tcpTutorialNotes.html