William Haack
Shawn Jain
Blake Elias

Design Doc

Brief Overview:

Our application is split into three parts: a server, a whiteboard picker client, and a whiteboard client. The whiteboard picker client is the portion of the application that lets users choose which whiteboard they want to work on and allows them to create new blank white boards for them and others to use. The white board client allows users to draw on a white board and broadcast their actions to the server for all the other white board clients to see.

Running the program:
Open the “whiteboard” project in Eclipse. Run the server from src/server/ServerRunner.java, and the client from src/canvas/WhiteboardPickerClient.java. Run the client from as many machines as possible. Make sure that the client code has src/shared/ConnectionDetails.java - variable SERVER_ADDRESS set to the IP address of the machine running the server.

After launching each client, follow the commands and buttons on screen to utilize whiteboard app.

Summary of Changes:
Between the time of our previous design and the final one, we have added some functionality and changed the architecture of scope that was retained.

We introduce a new addition to the client-server protocol that allows for a list of currently-connected usernames to be displayed next to each whiteboard when it is opened.

We have switched to a simpler client-server model which uses fewer levels of abstraction and fewer threads by relying on the network socket buffering abilities rather than our own threads to maintain queues of actions.

Data Types

Whiteboard: the underlying data structure for a whiteboard is a 2D array of pixels, each of which also has a color. We will use Java’s ImageBuffer to store this. There is a method called whiteboard.applyAction(Action a) - it takes an Action object (described below) and mutates the ImageBuffer stored in whiteboard to reflect this action.

WhiteboardAction: an action represents a segment that can be drawn between two points on the board. Specifically, it has int fields x1, y1, x2, y2 representing the coordinates of the endpoints of a segment, a Color field called color representing the color of this segment, and an

int field called strokeWidth representing the width of this pen stroke on the board.

Server:

WhiteboardServerInfo: (an individual whiteboard)

- clients (List<ClientConnection>) - all the clients connected to this whiteboard
- history (ArrayList<String>) - the list of all actions drawn on this whiteboard

ClientConnection: (connection between server and client)

- PrintWriter out
- String username

WhiteboardServer: (runs thread to receive and send messages for a particular whiteboard)

- List<WhiteboardServerInfo> whiteBoards -- all the whiteboards that exist on the server
- ServerSocket serverSocket -- the socket on which to accept new connections to this whiteboard from clients
- ArrayList<PrintWriter> clientOutputs -- all output streams to connected clients to this whiteboard
- runs thread which constantly listens for commands from any client that is connected to this whiteboard, and sends back messages to all clients that are connected. Messages include:
    - connection from user (update users list, send to all clients)
      protocol: "*newWhiteboardConnection whiteboardID username*"
    - add action (update master history in WhiteboardServerInfo, send new actions to all clients in the same format)
      protocol: "*addAction whiteboardID username x1 y1 x2 y2 colorRGB strokeWidth*"
    - disconnect action (close socket, send new users list to all remaining clients - see CanvasConnectionHandler message formats below)
      protocol: "*disconnectMe whiteboardID username*"

PickerServer: (sends list of all whiteboards on server, creates new whiteboards)

- List<WhiteboardServerInfo> whiteBoards -- all the whiteboards that exist on the server
- ServerSocket serverSocket -- the socket on which to accept new connections to this whiteboard from clients
- ArrayList<PrintWriter> clientOutputs -- all output streams to connected clients to this whiteboard
- runs thread which constantly listens for commands from any client, and sends

back messages to any or all clients. Messages:

- "*createNewWhiteboard*" creates a new whiteboard on the server and pushes an updated board list to all clients
- "*askForWhiteBoards*" sends a list of all whiteboards to all clients

The server maintains a few threads:

Communication with white board picker client:
The server has a thread that listens for messages that inquire how many whiteboards there are. This gives the whiteboard picker client a list of all the whiteboard available whiteboards, since whiteboards are represented by the integer corresponding to the order they were created in.
This same thread can also take a message that adds a new whiteboard to the list of whiteboards. Lastly, this thread can take an "open whiteboard" message which will spawn two new threads that handle the connection between the server and a whiteboard (i.e. the white board client that is simultaneously spawned on the user's machine).

Communication with white board client:
For each whiteboard, its thread liston the server.

- Opens a socket connection to each whiteboard client running on a user's machine. It constantly listens (and blocks execution while waiting) for messages containing WhiteboardActions. Whenever it receives an action it pushes it into whiteboard.history (must acquire a lock on whiteboard.history first, so as not to interfere with the updates in the second thread described below). (Where whiteboard is the WhiteboardServerInfo object for this board which is shared amongst all the threads communicating with clients connected to this whiteboard). This is what makes whiteboard.history have a complete history of every action drawn on the board.
- The thread runs a loop which constantly looks at the shared queue whiteboard.history,
  - sending all new[1] actions in this queue to the client (this is how one client hears about all other clients' changes - because all threads of the first type, described above, are pushing the actions they receive from their clients onto the history queue)
  - updating a variable, lastQueueSize, set to the value of whiteboard.history.size(), private to this thread
  - The above two operations must be executed together as one atomic operation, and will only run when they acquire a lock on

[1] The variable lastQueueSize is what allows the thread to send only new entries in whiteboard.history to the client. The only entries that need to be sent are indices in the range [lastQueueSize, whiteboard.history.size() - 1].

whiteboard.history. This is how we prove that no new actions are added from the first thread before the second thread updates its lastQueueSize, as such a condition would cause some actions to potentially not be sent to all clients.

User:

Whiteboard picker client:

When the user first opens our program, the program opens just a whiteboard picker client. When the whiteboard picker client starts it opens a connection to the server and displays all of the available whiteboards to edit. Upon double clicking a whiteboard a new window opens up with a white board client for the chosen board (a new thread, as described in the next paragraph). The user can open multiple whiteboards at a time by going back to the list of boards in the picker client - each will spawn its own whiteboard client. When the user clicks a button “create new whiteboard” the whiteboard picker client sends a message to the server in a new thread, and on response modifies the list of available whiteboards.

Whiteboard client (CanvasConnectionHandler):

The whiteboard client displays a given whiteboard and allows users to draw on it. The whiteboard client has a thread running that listens for user input and when the user makes these inputs it does both of the following:

- send actions to the server (which will be processed by a WhiteboardServer object -- for formats, see protocol in WhiteboardServer above)
- draws these actions on the Whiteboard on the screen (by calling whiteboard.apply(action) as well as gui.repaint() on the GUI object).

There is another thread running that receives and processes additional commands from the server. These include:

- “*addAction*”: (same format as that sent to server; see protocol in WhiteboardServer above) - draws an action on the board and repaints the GUI
- “*connectedUsers user1 user2 ...*”: set all of these usernames into the “current users” string
- “*yourUsernameIs username*”: informs the client what username the server saw it join as (i.e. after possibly being changed because of the name having spaces, being a duplicate, or being empty).

The above 2-thread system means a user’s actions will be drawn to his own screen immediately, but will also be sent to the server, and that the user’s board will eventually be updated (overwritten) with new, synchronized state from the server. This system has the property that all clients eventually will have the same picture on their screen if

provided enough time for all network transactions to complete.

The repaint is only once at the end of an iteration, rather than after every action, to give a slight speedup and to prevent a “flickering” effect as multiple actions are drawn that possibly overlap each other. If another user’s change really should have come before the local user’s change, but the local user’s change was drawn first because it was done locally, the other segment will be smoothly drawn “under” the user’s change once it is received over the network without any flickering (a “mine” - “theirs” - “mine” switcheroo would look annoying).

Testing Strategy:

We will have a testing strategy for each of the main components of our application.

Testing Server:

The server is considered working if we get the responses/behavior we expect after giving the server some mock messages from the client. We will test we get the expected responses/behavior from

- Creating a query for the list of white boards
- Sending a create new whiteboard query
- Sending a List<Action>

Testing whiteboard picker client:

The whiteboard picker client is considered working if it displays all the current whiteboard, gives the ability to create a new whiteboard, and allows users to open a new whiteboard. We will test we get the expected responses/behavior from

- Creating a new whiteboard
- Opening a whiteboard from the list.

We will also manually test to make sure all the visual components of the application are working.

Testing whiteboard client:

The whiteboard client is considered working if it allows the users to properly draw on the board, broadcasts those drawings to the server, and applies actions that are broadcasted from the server. We will test we get the expected response/behavior from

- Clicking any of the paintbrush modifying buttons
- Having a user draw on the board
- Having the server send a list of actions to apply

Like before, we will manually test to make sure all the visual components of the application are working.

Protocol:

MasterServer<->Whiteboard Picker Client (Master)

The whiteboard picker client can send the following two messages

1. Get all whiteboards
2. Create new whiteboards

The server responds to either message with a List<Integer>.

SlaveServer<->Whiteboard Client (Slave)

The whiteboard client only sends the following type of message:

List<Action>

The server also only sends a list of actions to the client (the actions that should be applied.)