6.005 Project 2 Design Doc

anandoza, jlrubin, mmou

Overview

This document covers the most significant decisions in our collaborative whiteboard project.

Major changes from our milestone design document regard our threads and Message types.

We restructured our threads such that we have 3n+1 threads for n connected clients. 1 thread handles the setup of connections, and 3 per user are used to give 1 for the User model to run, and 2 for I/O to the client. This is so that we can wait for read, write, and process data simultaneously. We use BlockingQueues to pass messages between threads.

In terms of Message types, we were originally going to "partially serialize" CurrentUser objects in order to send messages with updated lists of usernames to the client. Deciding that "partial serialization" was bad design since it breaks the fact that the object is truly serializable and deserializable, we created a new UserListMessage that would contain only a list of String usernames. Conveniently, we used generics on CurrentUser such that it's to/fromJSON methods return a UserListMessage. Another message decision was to split StrokeMessage into FromServerStrokeMessage and ToServerStrokeMessage. This way, we wouldn't be sending the same type of message from server and to server.

In our Client, we also simplified things greatly. We eliminated the local buffer because we felt they were overkill (and didn't do much to improve performance anyway).

In our final design doc, we also added a lot of additional Client GUI design detail.

User Interface

- How the whiteboard is structured
    - Main Board @Top
    - Tool Bar @Top
        - Whiteboard editing actions
            - Select color
            - Draw freehand
            - Erase
        - Whiteboard organizing actions
            - Switch whiteboards
                - New whiteboard
                - Connect to existing whiteboard
            - Change username
    - @Bottom
        - List of currently connected usernames

Testing Strategy

Considering the difficulty of automatically testing the GUI as well as concurrency issues, we use a combination of automatic unit tests and manual tests.

Overall, we wrote unit tests wherever it is practical. For example, we wrote unit tests for our JSONable, Drawable, SequentialIDGenerator classes. Especially for our JSONable message objects, it's important to ensure that objects that are serialized then deserialized are equivalent their original objects.

We wrote integration tests for the server to ensure that messages are getting passed back and forth as expected.

We wrote detailed testing strategies for classes dealing with the GUI, which can be found at the top of ConnectionGUI, DrawingGUI, and DrawingToolbar.

The specific cases we tested for in each class and method are further described in the sections below under the label "TESTING: ".

Threadsafe Argument

- Between Many Clients and Server
    - How whiteboards are named and accessed by users
        - By a randomly generated strings
    - How whiteboards are stored or cached (e.g. at a central server or on clients)

        - Whiteboards are stored with a list of Drawables
        - Strokes are assigned indexes in the whiteboard stack, and then are sent out with indexes to the clients which store a local copy of the whiteboard stack
            - Local whiteboards should cache their own strokes and lock them into the shared state once an acknowledgement is made
    - What guarantees are made about the effects of concurrent edits
        - Edits will be stored as happenings between mousedown and mouseup and will be assigned a layering order by the server, which will then be synced to all clients (including the creator)
        - Truly 'simultaneous' edits will be resolved arbitrarily, but will be synced across clients.
- On the Client
    - All GUI operations will be done on event-dispatch thread
    - Model operations will be handled on a separate thread (SwingWorker with publish/process)
    - One thread for writing to server
    - One thread for reading from server
- On the server
    - There is exactly one thread for each client, which has a blocking loop to receive client messages, handle them, and send messages to the client.
    - Handling messages to/from clients will be centrally synchronized (using a lock) to avoid interleaving.

Server/Client Protocol

We will send messages between the server and client using JSON for steez (style+ease). Therefore we do not need to define a separate grammar, as our protocol is essentially made up of all the fields in our Message objects.

Our list of message types is below; they are further described under "Messages."

Client to Server:
NewWhiteboardMessage
SwitchWhiteboardMessage
SetUsernameMessage
ToServerWhiteboardMessage

Server to Client:
WhiteboardCreatedMessage
FromServerStrokMessage
UserListMessage

Design:

- ## Whiteboard Models
    - **Class WhiteboardServerModel**
        - Instance variables:
            - **public final String id**: unique ID per Whiteboard instance, a random String
            - **private List<Drawable> drawablesList**: sequence of Drawables
                - Note: elements with smaller indices are considered to be added earlier (so should be covered by elements with larger indices)
                - clients messages are appended
            - **private CurrentUsers users**
                - the currently connected clients should be maintained here
            - **private SequentialIDGenerator sequentialIDGenerator**

        - Generator of sequential message IDs.
    - Methods
        - **public WhiteboardServerModel():**
            - Constructor
        - **public synchronized void handleDrawable(Drawable drawable)**:
            - append drawable to end of drawablesList and then create a strokeMessage to sync clients with.
            - TESTING:
                - Test that drawablesList is updated correctly.
                - Test that the created strokeMessage is correct.
        - **public void broadcastStroke(StrokeMessage m)**
            - send all clients the StrokeMessage
            - TESTING:
                - Test that broadcastStroke is successful by testing that the clients handle the message correctly (see WhiteboardServer.handleRequest)
        - **public void addClient()**
            - add a client and then send them all the data needed
            - TESTING:
                - Test that clients variable is updated, and also that the client receives all the data successfully.
        - **public void removeClient(User user)**
            - Remove a user to currentUsers users if it exists.
- **public class User implements JSONable<User>**
    - instance variables
        - **public static final String KILL_MSG = "";**
        - **private static final String DO_NOTHING = "Donot";**
        - **private String username;**
        - **private final BlockingQueue<String> inQueue;**
        - **public final BlockingQueue<String> outQueue;**
        - **private WhiteboardServerModel wb;**
        - **private final ConnectedUser connection;**
    - methods
        - **public void input(String msg)**
            - Add a control message to be processed by the user in User's run method.
        - **public void output(String msg)**
            - Add a control message that's a String to be processed by the server in ConnectedUser's WriterQueue.
        - **public void run()**
            - takes in messages from inQueue
        - **public void handleRequest(String input)**
            - handles messages
        - **public void setName(String newName)**
            - Set instance's username.
- **public class CurrentUsers**
    - instance variables
        - **private List<User> users**
            - All currently logged in users
        - **private final String whiteboardID**
    - methods
        - **public void broadcast(String msg)**
            - send messages to all CurrentUsers
        - **public void addUser(String name)**
            - Update users accordingly if name is not already in users. Else don't do anything.
            - TESTING:
                - Test that the user is successfully added to users variable if the user name is unique.

                - Test that if the user name is already taken, the users variable is not updated (should alert the client somehow).
            - **public void removeUser(String name)**
                - Update users accordingly if name is in users. Else don't do anything.
                - TESTING:
                    - Test that the user is successfully removed from the users variable if it exists.
                    - Test that nothing happens if the user name doesn't exist in users.
            - **public synchronized void broadcastSelf()**
                - Send all users a UsersListMessage (usually after add/remove user).
                - TESTING:
                    - Test that all WhiteboardServerModels update their users variables correctly.
            - CurrentUser's JSONable methods deal with UserListMessage types, which take a list of only the String usernames of the Users in CurrentUser.
    - **Interface Drawable**
        - Method
            - **public void draw()**
                - Draw the object in the GUI
                - TESTING:
                    - Test that the correct figure is drawn (manual)
    - **public class DrawablePath implements Drawable, JSONable<DrawablePath>**
        - Instance variables:
            - **private final ArrayList<DrawableSegment> segmentsList**
        - Methods:
            - **@Override public JSON toJSON()**
            - **@Override public DrawablePath fromJSON(String data)**
            - **@Override public void draw()**
                - Calls draw() on each of the DrawableSegments
    - **public class DrawableSegment implements Drawable, JSONable<DrawableSegment>**
        - Instance variables:
            - **private final x1:** x coordinate of starting point
            - **private final y1:** y coordinate of starting point
            - **private final x2:** x coordinate of ending point
            - **private final y2:** y coordinate of ending point
        - Methods:
            - **@Override public JSON toJSON()**
            - **@Override public DrawableSegment fromJSON(String data)**
            - **@Override public void draw()**
                - Base case of RDT
                    - Canvas.drawLineSegment
                - Draw a line between two points (x1, y1) and (x2, y2), specified in pixels relative to the upper-left corner of the drawing buffer.

- # Messages
    - **Interface JSONable<T extends JSONable<T>>**
        - Interface that allows JSON serialization and de-serialization.
        - Method
            - **public T fromJSON(String jsonString)**
                - Convert a string to the original object type. (passes to fromJSON(JSONObject j). Caller must check that the String can be de-serialzed to this type.
            - **public T fromJSON(JSONObject j)**
                - Convert a JSONObject to the <T> object type. Caller must check that the String can be de-serialzed to this type.
            - **public JSONObject toJSON()**
                - Convert the original object into a JSONObject.
        - TESTING:

            - Test that each class (below) that implements jSONable, converted to JSON, is able to be converted back to the same object
    - **public class NewWhiteboardMessage implements JSONable<NewWhiteboardMessage>**
        - NewWhiteboardMessage is sent only from Client to Server, when the client would like to connect to a new whiteboard without regard for the whiteboard name. The server should respond with a SwitchWhiteboardMessage. This or SwitchWhiteboardMessage should be sent by client before sending any StrokeMessages.
    - **public class WhiteboardCreatedMessage implements JSONable<WhiteboardCreatedMessage>**
        - A message to send when a new whiteboard gets created. (should be treated similarly to a SwitchWhiteboardMessage on client)
        - JSON object contains:
            - name: whiteboardID
    - **public class UserListMessage implements JSONable<UserListMessage>**
        - This message should be sent whenever the users/usernames on a whiteboard are updated.
        - JSON object contains:
            - values: list of users
            - whiteboardID: whiteboardID
    - **public class SetUsernameMessage implements JSONable<SetUsernameMessage>**
        - The client should send this Message to the server when it wants to update its name.
        - JSON object contains:
            - id: username
    - **public class SwitchWhiteboardMessage implements JSONable<SwitchWhiteboardMessage>**
        - SwitchWhiteboardMessage is sent both from Client and Server. When the client would like to connect to a specific whiteboard or create a new one with that whiteboard name, this message should be used. When the server connects a user to a specific whiteboard, this should be sent to the client with the new name. This or NewWhiteboardMessage should be sent by client before sending any StrokeMessages.
        - JSON object contains:
            - id: whiteboard ID
    - **public class ToServerStrokeMessage extends StrokeMessage**
        - Sub-class of StrokeMessage used so that client implementors can create StrokeMessages with only the appropriate fields, as well as more convenient naming.
    - **public class FromServerStrokeMessage extends StrokeMessage**
        - Sub-class of StrokeMessage used so that client implementors can create StrokeMessages for sending to the Client with only the appropriate fields, as well as more convenient naming.
    - **public class StrokeMessage implements JSONable<StrokeMessage>**
        - This message should be used to communicate when something is drawn. It shouldn't really be constructed directly, although it can. (better to use FromServerStrokeMessage and ToServerStrokeMessage)
        - This message type contains all the information needed to integrate a message on the server or client. Information not needed (ie, id on incoming message) may not be accurate).
        - Use getDeleteMessage to create a DeleteStrokeMessage corresponding to this message.
        - Use withServerID to return a StrokeMessage with the appropriately set serverID.
        - JSON object contains:
            - id: this.id
            - userSeqId: this.userSeqId
            - drawable: this.drawable.toJSON()
            - username: this.username
            - wb: this.whiteboardID

- # Server
    - **class WhiteboardServer**
        - Instance variables
            - **HashMap<String, WhiteboardServerModel> openWhiteboards**
                - hashmap of whiteboards, each of which stores that whiteboard's "master" list of Drawables from a string key.
        - Methods
            - **public static void main(String[] args)**
                - parses command-line argument; if successful, calls runWhiteboardServer()
                - TESTING:

                - Test that args configure server properly (responds to different port #s)
                - Test that server is actually running.
            - **public WhiteboardServerModel createWhiteboard()**
                - create new whiteboard
            - **public static void runWhiteboardServer(int port) throws IOException**
                - Start a WhiteboardServer running on the specified port
            - **public static void serve()**
                - Run the server, listening for client connections and handling them in separate threads by calling handleConnection(socket)
                - TESTING:
                    - Test that packets get sent to the proper handler
            - **private void handleConnection(Socket socket)**
                - Handle a single client connection, calling handleRequest to parse client inputs. Returns when client disconnects.
                - TESTING:
                    - Test that when passed a mock socket, the proper actions are taken.
    - **public ConnectedUser**
        - Starts 3 threads:
            - ReaderThread
                - BufferedReader takes in messages and puts them in user's inQueue
            - WriterThread
                - Takes from user's outQueue and prints messages
            - User Thread
                - See User class

- # Client
    - **class ClientGUI** - handles running the client
        - **public static void main(String[] args)**
            - run client by starting here
            - creates new ConnectionGUI
        - **public void connect(String ip, String port)**
            - creates new DrawingGUI
            - Precondition: ip and port must be valid IP addresses and port numbers, respectively
    - **class ConnectionGUI**
        - Creates connection GUI that asks for a user's IP address and port number.
        - TESTING:
            - Default values: IP address should default to 127.0.0.1, port number to 4444.
            - Pressing 'connect' button: If IP and port are valid, ConnectionGUI should close, and a DrawingGUI should start up.
    - **class DrawingGUI**
        - Creates drawing canvas GUI that includes the canvas and the drawing tools.
        - Includes DrawingToolbar
        - TESTING:
            - Verify that all data shows OK without lag, ie test usability
            - Test logging in to server running on several ports/various IP addresses.
            - Test connecting to a whiteboard which already exists and one which does
            - not
            - Test creating a new whiteboard
            - Test Switching brush sizes
            - Test lag with three open clients (how long to display)
            - Test that names update appropriately with three clients
            - Test Eraser works
            - Test changing color
            - Test changing color while on Eraser keeps it as an eraser.
            - Test That userlist shrinks when a client disconnects/switches boards.

            - Test changing usernames
    - **class DrawingToolbar**
        - Creates drawing toolbar GUI.
        - Includes: pen/eraser chooser, color picker, change username button.
        - TESTING:
            - Clicking pen radio button: only pen radio button selected, user should be able to draw black lines
            - Clicking eraser radio button: only eraser radio button selected, user should be able to erase by drawing thick white lines
            - Clicking color button: should bring up a color selector window. After user updates the color, the button should be the selected color, and the user should be drawing in the selected color.
            - Clicking username button: should bring up a window to input a new username. Should allow any username that's not an empty string to be selected. Change should be reflected in "current user" list at the bottom of the DrawingGUI.
    - **class WhiteboardClient**
        - **Instance variables**
            - **public final WhiteboardClientModel model**
            - **public final String whiteboardID**
            - **private Map<Integer, Drawable> serverDrawables = new HashMap<Integer, Drawable>();**
        - **Methods**
            - **private <T> List<T> getSortedListFromMap(Map<Integer, T> map)**
            - **public void draw(Drawable drawable)**
            - **public void addDrawableFromServer(int id, Drawable drawable)**
            - **public Image getImage()**
    - **class WhiteboardClientModel - COPIED AND PASTED FROM MILESTONE!!!**
        - Instance variables
            - **private Color color = Color.BLACK**
            - **private int brushWidth = 5**
            - **private DrawingTool tool**
            - **public BlockingQueue<String> outgoing = new LinkedBlockingQueue<>()**
            - **public BlockingQueue<String> incoming = new LinkedBlockingQueue<>()**
            - **public Whiteboard whiteboard**
            - **private String username**
        - Methods
            - **public void setTool(Class<? extends DrawingTool> toolClass)**
            - **public void sendMessage(JSONable message)**
                - add a message to outgoing queue
            - **public void switchWhiteboard(String whiteboardID)**
                - send either NewWhiteboardMessage or SwitchWhiteboardMessage

- **Other**
    - **global**
        - Stores some constants.
    - **utils**
        - **public class SequentialIDGenerator**
            - Generates sequential integer ID's. Each object that needs to issue unique sequential ID's should have its own instance.