6.005 Project 2 Milestone Design Doc
anandoza, jlrubin, mmou

Overview

This document covers the most significant decisions in our collaborative whiteboard project. Some of the choices (such as using arrays v.s. Lists) made here may not be reflected in the final implementation. Additionally, we have some optional fields which we are considering as extra features, as such we have not written testing strategy for them nor applied the same amount of attention to detail as in the main corpus of this document. However the required and other major design choices in architecture are accurately represented.

User Interface

- How the whiteboard is structured
    - Main Board @Top
    - Tool Bar @Bottom
        - Whiteboard organizing actions
            - Connect to existing whiteboard
            - New whiteboard
            - Switch between whiteboards
            - See list of currently connected usernames
            - Optional:
                - Recent History tab switch
        - Whiteboard Editing Actions
            - Select color
            - Draw freehand
            - Erase
            - Optional:
                - Playback
                - Clear all
                - Select brush width
                - Pick Shapes
        - Optional:
            - Save screenshot

Testing Strategy

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

Overall, we will write unit tests wherever it is practical, for example, for most of our methods that deal with the JSONable message objects. Other actions will be unreasonable to unit test, if, for example, they rely on the server sending something successfully and the client receiving something successfully. We will write unit tests for both the server and client methods, then write integration tests to test the interactions between components. To test for concurrency issues, we will manually test. Also we will manually test the methods that interact with the GUI to ensure that the GUI looks correct.

Specific tests for each class and method are 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

Server to Client:
WhiteboardCreatedMessage
CurrentUsers

Bidirectional:
StrokeMessage

Design:

- ## Whiteboard Models
  - **Class WhiteboardClientModel**
    - Instance variables
      - **private final Map syncedState<Integer, Drawable>**
        - collection of Drawables that have been acknowledged by the server and are now displayed on the client GUI
      - **private final Map localState<Integer, Drawable>**
        - collection of Drawables which haven't been acknowledged by the server yet
    - Methods
      - **public Image displayBuffer()**
        - return the current contents of the buffer (to be called after all objects in the buffer have been drawn) as an Image
        - TESTING:
          - Test that all drawn objects in the buffer are shown
      - **public void drawBuffer()**
        - render what the buffer should (not will immediately) show in background
        - Synchronized on something so that they are done in proper order
        - process synced then local state
        - TESTING:
          - Test that objects are being drawn in the buffer are not yet visible to user
          - Test that objects are drawn in the correct order
      - **public void addDrawableAt(int syncedTo, int local, Drawable drawable)**
        - The global state has assigned a certain index for the board, so add it to syncedState and remove from localState
        - TESTING:
          - Test that the Drawable is successfully added in syncedState and removed from localState
    - Optional:
      - **private int[][] baseDrawing**
      - **public void cacheCompleted(int to)**
        - cache the full sequence syncedState rendering in baseDrawing so we don't always have to re draw all of it.
      - **public void detectMissing()**
        - search the syncedState to see if we may be missing any messages and initiate a request for retransmission
  - **Class WhiteboardServerModel**
    - Instance variables:
      - **public final String id**: unique ID per Whiteboard instance generated from the random combination of 2 or more words from a large dictionary (i.e., correct horse battery staple).
      - **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 Map<String, String> clients**
        - Should keep track of its clients, assigning them their own ID's and letting them set usernames

        - **private CurrentUsers users**
            - the currently connected clients should be maintained here
    - Methods
        - **public WhiteboardServerModel(String id):**
            - Constructor
        - **public synchronized void addDrawable(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 class User implements JSONable<User>**
    - instance variables
        - **private final String username**
        - **private final Socket socket**
        - **private final BlockingQueue queue**
    - methods
        - **public void add(String msg)**
            - add a msg to the queue to be sent out
        - **public void start()**
            - start the User thread
- **public class CurrentUsers**
    - instance variables
        - **private final Map<String, Socket> users**
            - All currently logged in users
        - **private final String timestamp**
            - a monotonically increasing field in case msgs arrive out of order
        - **private final String whiteboardID**
    - methods
        - **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 broadcast(String msg)**
            - send all msgs to CurrentUsers
        - **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 void broadcastSelf()**
        - Send all users a CurrentUsersMessage after add/remove user
        - Sync on users
        - TESTING:
            - Test that all WhiteboardServerModels update their users variables correctly.
    - **public JSON toJSON()**
        - Only handles names of users
- **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>**
    - 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>>**
    - Method
        - **public JSON toJSON()**
            - convert data to a json type
        - **public T fromJSON(String data)**
            - convert data back from a string
    - TESTING:
        - Test that each class (below) that implements jSONable, converted to JSON, is able to be converted back to the same object

- **public class StrokeMessage implements JSONable<StrokeMessage>**
  - Instance variables:
    - **Private final int id**
      - unique id for the StrokeMessage generated sequentially by server
    - **private final int userSeqId**
      - unique id for the StrokeMessage generated sequentially by user (which is where they store that in their buffer)
    - **Private final Drawable drawable**
    - **Private final String username**
      - client username that drew the Drawable
    - **Private final String whiteboardID**
      - id of WhiteboardModel
    - **Private final Color color**
      - from http://docs.oracle.com/javase/7/docs/api/java/awt/Color.html
    - **Private final int brushWidth**
      - represents thickness of brush strokes
  - Methods:
    - **public StrokeMessage(Drawable drawable, int id, String username, String whiteboardId, Color color, int brushWidth)**
    - **@Override public JSON toJSON()**
    - **@Override public StrokeMessage fromJSON(String data)**
    - **public Drawable toDrawable()**
- **public class NewWhiteboardMessage implements JSONable<NewWhiteboardMessage>**
  - methods
    - **@Override public JSON toJSON()**
    - **@Override public NewWhiteboardMessage fromJSON(String data)**
- **public class WhiteboardCreatedMessage implements JSONable<WhiteboardCreatedMessage>**
  - instance variables
    - private final String whiteboardID
  - methods
    - **@Override public JSON toJSON()**
    - **@Override public NewWhiteboardMessage fromJSON(String data)**
- **public class SwitchWhiteboardMessage implements JSONable<NewWhiteboardMessage>**
  - instance variables
    - **private final String whiteboardID**
  - methods
    - **@Override public JSON toJSON()**
    - **@Override public NewWhiteboardMessage fromJSON(String data)**

Server

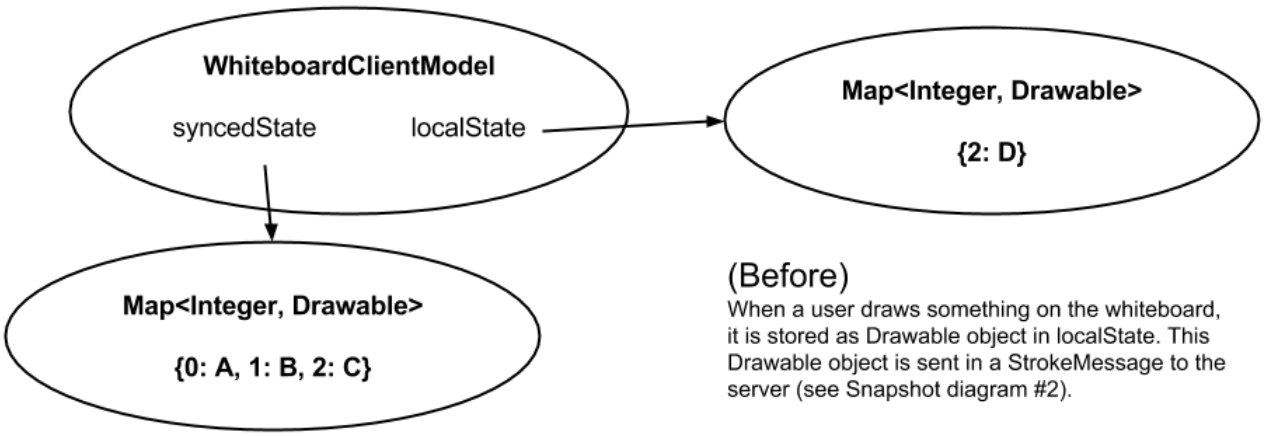
- **class WhiteboardServer**
  - Instance variables
    - **HashMap<String, WhiteboardServerModel> openWhiteboards**
      - hashmap of whiteboards, each of which stores that whiteboard's "master" list of Drawables from a string key.
  - Methods
    - **public static void main(String[] args)**
      - 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 static void runWhiteboardServer()**
        - Run the server, listening for client connections and handling them in separate threads by calling han
        - dleConnection(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.
    - **private String handleRequest(String input)**
        - Parses a client's input, performing requested operations and returning the appropriate message.
        - Allows client to connect to and create specific whiteboards using the protocol (see below).
            - If id is new, create a new WhiteboardModel
            - Else if id is already used, serves entire sequence of Drawables that exist for that WhiteboardModel upon connection.
        - TESTING:
            - Test that responds with appropriate message given a packet
            - Test If id is new, create a new WhiteboardModel
            - Test if id is already used, serves entire sequence of Drawables that exist for that WhiteboardModel upon connection.

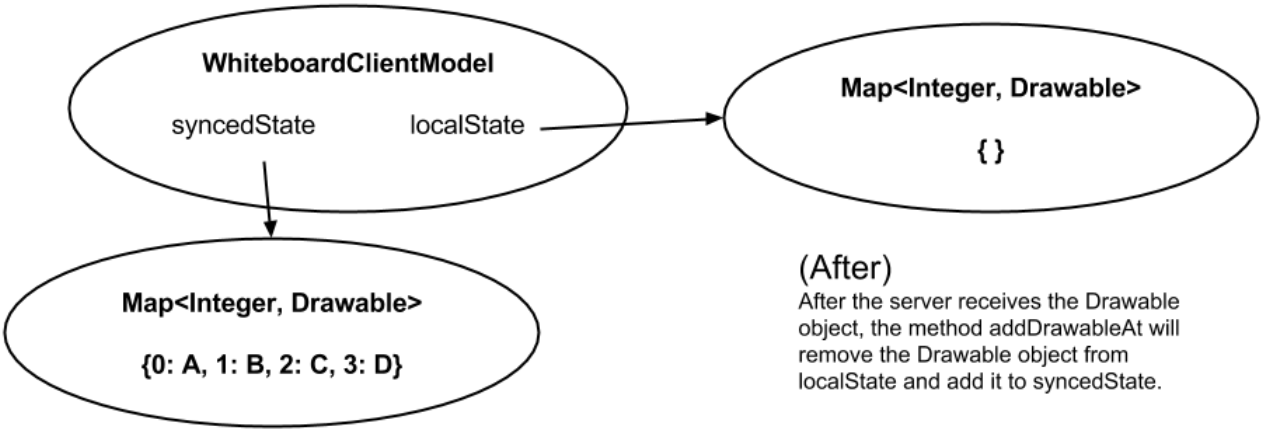
- **Client**
    - **class WhiteboardClient**
        - instance variables
            - **private String address**
                - the server to connect to
            - **private String username**
                - username of client
            - **private String whiteboardID**
                - id of WhiteboardModel it connects to
            - **public WhiteboardClientModel model**
                - the whiteboard client side model
            - **private final BlockingQueue<Runnable> queue**
                - Actions get sent here to be executed in order.
                - added to by handleMessage
        - Methods
            - **public void runClient()**
                - run the client in a background thread
                - when a message is in the queue, execute its runnable which should have all actions needed packaged in.
                - TESTING:
                    - Test that client runs
            - **public StrokeMessage mousePathToMessage(int[][] mousePath)**

        - convert a mouse path (assuming it is expressed as a 2-d array of points) to a stroke message with a unique sequential id.
        - TESTING:
            - Test that a mouse path is properly converted to a Stroke
    - **public void handleMessage(String message)**
        - respond appropriately to a msg by converting to JSON then to appropriate type, then adding its runnable to queue which will perform any necessary updates to GUI/Server
        - TESTING:
            - Test the insertion order of messages sent in with odd orders from server
    - **public void newWhiteboard()**
        - Generate a new whiteboard and switch to it.
        - TESTING:
            - Test that whiteboard is new/does not affect data of other existing whiteboards.
            - Test that no data from last whiteboard creeps into this one (through a late response from server)
    - **public void switchWhiteboard(String id)**
        - switch to a given whiteboard
            - could handle things liked cached copies etc
            - request all Drawables for that board already existing
        - create a new whiteboard with that name
        - TESTING:
            - Test that all whiteboard data is received
            - Test that no data from last whiteboard creeps into this one (through a late response from server)
            - Test if no whiteboard exists, create a new one with that id
    - **public boolean changeName(String name)**
        - change username
        - return false if name taken
        - TESTING:
            - Test that username updates properly when unique
            - Test that non-unique username change fails
- **class WhiteboardGUI**
    - instance variables
        - **private WhiteboardClient client**
    - Methods:
        - **public WhiteboardGUI()**
            - construct a new whiteboard
        - **public void runGUI()**:
            - serve up a GUI for this WhiteboardModel that loads the sequence of Drawables that exist upon connection
        - **public void reset()**
            - run a new client.
    - TESTING:
        - Verify that all data shows OK without lag, ie test usability
        - Verify that usernames shows all editing clients

Snapshot Diagram #1
What happens when shape "D" is sent to the server, synced, and sent back to the client.

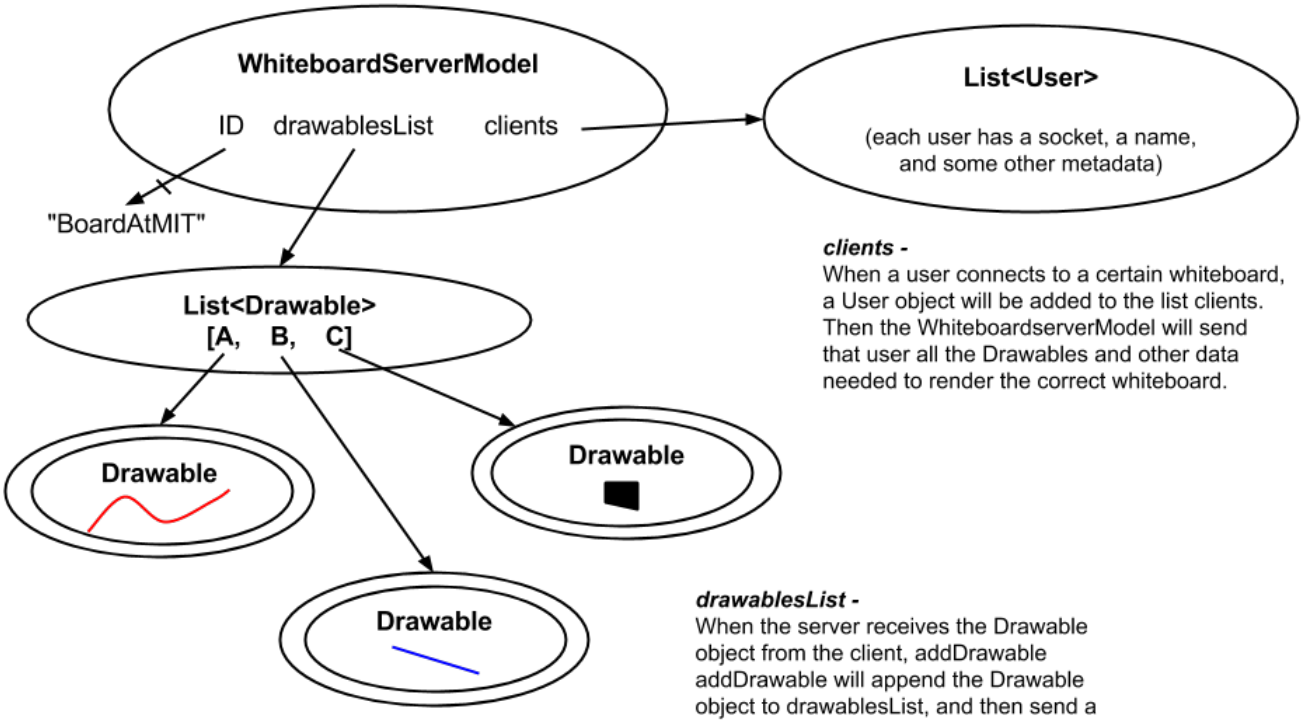


(Before)
When a user draws something on the whiteboard, it is stored as Drawable object in localState. This Drawable object is sent in a StrokeMessage to the server (see Snapshot diagram #2).



(After)
After the server receives the Drawable object, the method addDrawableAt will remove the Drawable object from localState and add it to syncedState.

Snapshot Diagram #2
A snapshot diagram of WhiteboardServerModel.



clients -
When a user connects to a certain whiteboard, a User object will be added to the list clients. Then the WhiteboardserverModel will send that user all the Drawables and other data needed to render the correct whiteboard.

drawablesList -
When the server receives the Drawable object from the client, addDrawable addDrawable will append the Drawable object to drawablesList, and then send a strokeMessage back to the client.

(A "Drawable" is an interface that encapsulates something that can be drawn. For example, a line segment of a particular color.)