6.005 Project 2 Design Document (Revised)

Eric Ruleman, Katherine Yu, Catherine Yun

December 8, 2013

1 Overview/Implementation Choices

Our implementation of the collaborative whiteboard currently allows users to draw ovals and make freehand marks. We also allow users to undo and redo changes to each whiteboard. Each client has a reference to its own model that stores the state of a given whiteboard, but these models are not shared between clients, or client and the server. They are each altered the same way (given that they all pertain to the same whiteboard) through message passing and a shared edits queue in the server. All edits must be processed by the server before they are populated on a client's canvas, ensuring that no concurrency bugs occur.

Each whiteboard is identified by a numerical integer ID number. Users can access a specific whiteboard from the whiteboard selection pane upon first entry, or they can switch between whiteboards once they have already entered a whiteboard.

2 Snapshot Diagram
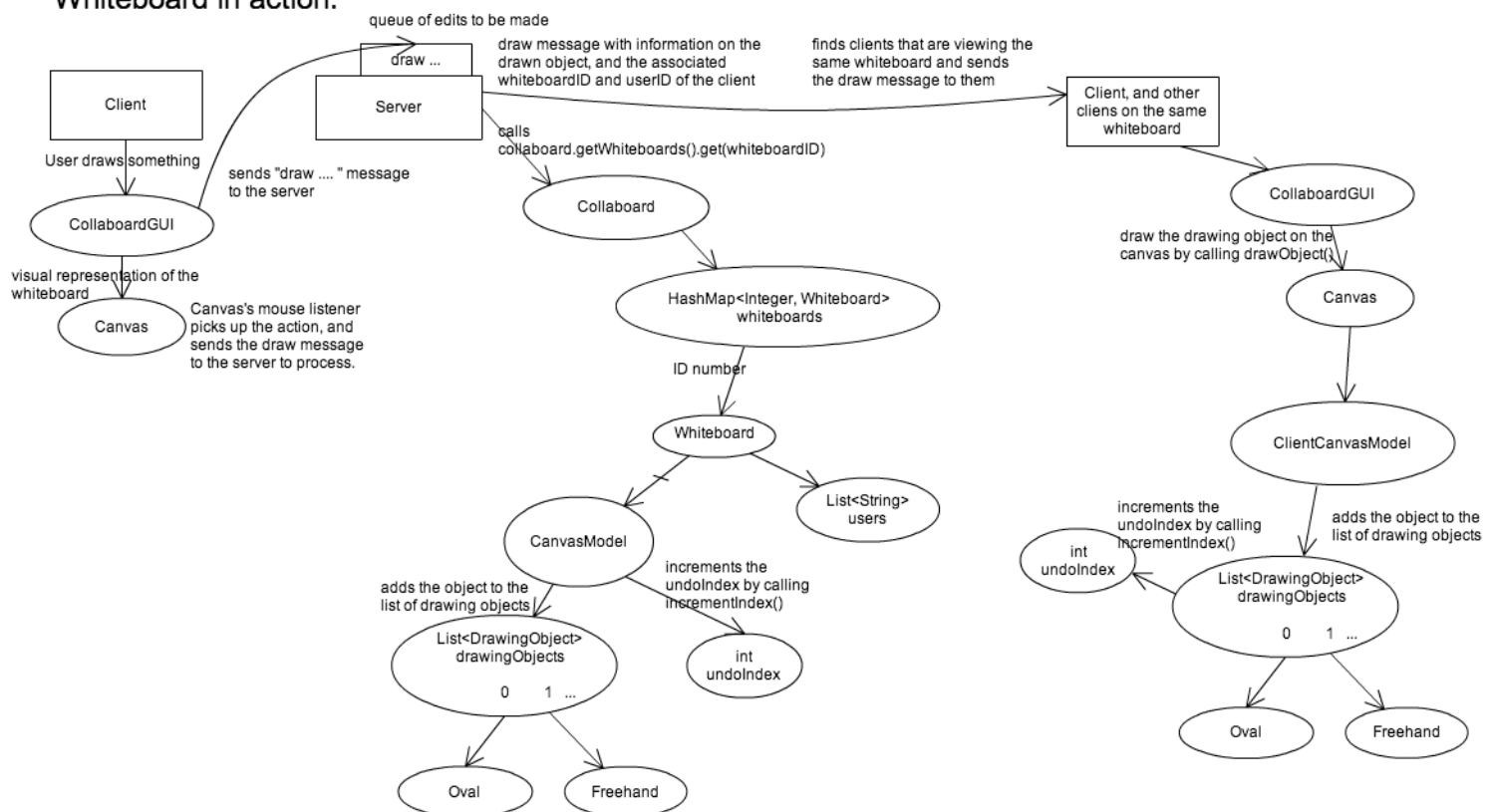
3 Datatype Design

CollaboardServer: Server class for our collaborative whiteboard.

- Stores the current instance of Collaboard, and current active UserThreads.

- Handles messages passed from each client thread.

- Has a BlockingQueue for requests to mutate a whiteboard to be put in and handled one at a time by a thread specifically dedicated to handling requests

Collaboard: Class that stores all the active whiteboards and active usernames during a given server session.

- has a Set of usernames in order to prevent users from creating duplicate usernames

- has a HashMap of WhiteboardIDs to whiteboards to prevent users from creating whiteboards with duplicate ID numbers

Whiteboard: Class representing the state of a given collaborative whiteboard.

- has a CanvasModel

- has a list of users currently accessing that whiteboard

- has a whiteboardID number

- threadsafe through use of the monitor pattern (will elaborate later in concurrency strategy)

CanvasModel: Class representing the state of the canvas in a whiteboard.

- stores an ArrayList of DrawingObjects in the order that they are drawn on the canvas.

- stores an undoIndex to determine which objects should be displayed on the canvas

- threadsafe through use of the monitor pattern (will elaborate later in concurrency strategy)

DrawingObject: An Interface implemented currently by Freehand and Oval.

- Each DrawingObject represents a new stroke or image added to a canvas.

- This makes it easy to add new features to our whiteboard, like the ability to draw other shapes like triangles or rectangles.

- Freehand: represents a freehand stroke made by a user. Contains a list of Line objects, which store a starting x and y coordinate, an ending x and y coordinate, a string representation of a color, and a string representation of a thickness.

- Oval: represents an oval made by a user. Contains a starting x and y coordinate, a center x and y coordinate, a string representation of a color, and a string representation of a thickness.

User: Class representing a user currently using the collaborative whiteboard system.

- has a userID - unique and assigned by the server upon connection. Used for identification purposes. (Also allows our code to be modified in the case that we allow users to change their usernames without disconnecting.)

- has a username: name that is displayed when the user enters a whiteboard.

- has a whiteboardID - ID of the Whiteboard the user is currently accessing.

- has a ToolbarModel: Class representing the Color and Thickness settings a given user has active.

UserThread: Class representing a client connection on the server.

- stores a username, userID, and currentWhiteboardID.

Client: Class representing a socket connection to the server. Also handles inputs from the server.

- stores a User object

- stores a reference to a CollaboardGUI in order to call methods accordingly to mutate the GUI according to inputs from the server.

- all of these such calls are made using SwingUtilities.invokeLater() to prevent concurrency issues with Swing.

CollaboardGUI: GUI class of the collaborative whiteboard. Populated upon connection to the server.

- The first panel the client sees is one in which they select a username. If the username is taken or invalid, an error is displayed. Otherwise, the username is stored client-side and by the server and the user is taken to the next window.

- The second panel the client sees is for whiteboard selection. The client can either choose to create a new whiteboard, or select an existing whiteboard from the list of existing whiteboard IDs.

- Once a valid whiteboard is created or selected, the main GUI is shown, consisting of a Canvas, a ToolbarGUI, a JScrollPane displaying active users, and a header that allows the user to switch whiteboards by inputting a new whiteboardID.

- ToolbarGUI: Has buttons that the users can toggle to select a drawing color and thickness of the user's ToolbarModel, as well as undo and redo changes to the Canvas.

- has a ClientCanvasModel (in the Canvas) - each client has its version of the CanvasModel stored by the server.

4 Protocol

From the Server to the Client:

- **VALIDUSER** Allows the client to proceed to the whiteboard selection pane

- **VALIDWHITEBOARD** Allows the client to proceed to the specified whiteboard

- **USERTAKEN** The username the client selected is invalid. Prompts an error message on the username selection pane.

- **WHITEBOARDTAKEN** The whiteboardID the client tried to create a whiteboard with is taken. prompts an error message.

- **READY** The canvas is ready to be initialized

- **INITDRAW** Sent when a client enters a new whiteboard. Encodes a DrawingObject for the client to add to its ClientCanvasModel.

- **UNDOINDEX** Sent when a client enters a new whiteboard. encodes the undoIndex of the Whiteboard's CanvasModel. Prompts the client's GUI to be populated with DrawingObjects up until the undoIndex.

- **DRAW** Encodes a DrawingObject to be added to the client's ClientCanvasModel and to be populated on the Canvas

- **ENTER** Contains the username of a client that has just entered the same whiteboard the current client is using. Prompts the GUI to update its list of users accordingly.

- **EXIT** Contains the username of a client that has just exited the whiteboard the current client is using. Prompts the GUI to update its list of users accordingly.

- **UNDO** Prompts the GUI's canvas to perform the undo action.

- **REDO** Prompts the GUI's canvas to perform the redo action.

From the Client to the Server:

- **MAKEBOARD** Desired whiteboardID. Checked for duplicates.

- **MAKEUSER** Desired username of the client. Checked for duplicates.

- **SWITCH** Client wants to switch whiteboards.

- **WHITEBOARDTAKEN** The whiteboardID the client tried to create a whiteboard with is taken. prompts an error message. Changes to a specific whiteboard - The server finds threads that are accessing the affected whiteboard and passes the messages back to them:

- **UNDO** Client wants to undo a change. Put on the shared BlockingQueue to be handled.

- **REDO** Client wants to redo a change. Put on the shared BlockingQueue to be handled.

- **DRAW** Client made a change to the canvas. Put on the shared BlockingQueue to be handled.

- **ENTER** Client is entering a new Whiteboard. Adds the client from the Whiteboard's list of active users.

- **EXIT** Client is exiting a Whiteboard. Removes the client from the Whiteboard's list of active users.

5 Concurrency Strategy

We will have a shared message blocking queue for the server. All edits will be stored in the queue to be handled by a dedicated edits-handling thread in order. Therefore clashing edits cannot occur. draw, undo, and redo messages are put in this queue, since they have the potential to clash. All other messages are handled in each clients UserThread.

CanvasModel, Whiteboard, and Collaboard, whose information is stored on the server, are threadsafe through use of the monitor pattern, to ensure that multiple threads wont mutate them concurrently, and cause edits to be lost, and to ensure that they wont view any of these objects while they are being mutated. Accesses to these objects occur on the server when the user first connects (the server will access collaboards list of users and active whiteboards and send the appropriate messages to the client), and when the user connects to a specific whiteboard (the server will send the user a list of the Whiteboards list of active users, its CanvasModels undoIndex, and its list of DrawingObjects). Since these methods are all observing the state of these objects and not mutating them, it is sufficient to just use locking to achieve thread safety. Deadlock will not occur, because if a particular request requires access to more than one of these objects, they will always obtain references/locks to these objects in the order: Collaboard, Whiteboard, CanvasModel.

On the client side, all edits to the GUI are made in the event handling thread through the use of SwingUtilities.invokeLater(). All client threads have their own instance of Client-CanvasModel, which ensures thread safety through confinement.

6 Testing Strategy

We have 3 levels of testing: 1. Testing our abstract data types. 2. Integrated Tests 3. GUI Testing *Flesh this one out please*