Design Document

Project 2 – Collaborative Whiteboard

Milestone 1 – December 3rd, 2013

Genghis Chau, Kevin Peng, Parker Zhao

Datatype Design

The first datatype that we will implement will be the whiteboard, which will be represented mostly with a Canvas object. The Canvas will contain much of the information about the drawing, including what strokes have been made on the whiteboard, which pixels are colored in and with what color, and the general structure of the drawing. The whiteboard also contains a String representing the name given to it, which will be decided on creation. The name of the whiteboard can be modified later as well, given that the new name does not conflict with another existing whiteboard. These names will be unique amongst whiteboards currently active on the server, and will be used to identify them. Along with its name, the whiteboard will also keep track of the current users accessing the board, which will be used to identify who is editing and viewing the whiteboard. We will use a hash map to make sure that users do not choose the same username. Note that users will not be able to resize a whiteboard once it is created on the server.

Along with the whiteboard, we will also have a User/Client datatype that represents an individual using the whiteboard. The user will have several attributes including a username, pen thickness, pen color, and a whiteboard. The username is used as an identifier; no User can have the same username as another User on the same whiteboard. The whiteboard will have a list of usernames corresponding to all Users connected to it. The pen thickness and pen color are chosen by the user using the GUI and will be sent to the server when new strokes are made to the whiteboard using the protocol (see *Protocol*). The whiteboard is the User's personal model of the shared whiteboard in which local changes will be made first before sending to the server.

These two datatypes will also have mutator and observer methods, which are detailed in the general datatype descriptions below.

The user will have access to his own version of the shared whiteboard, where changes will first be made. The user's actions will call mutator methods to the whiteboard, which will then be sent using an HTTP protocol to the server version of the whiteboard. The server will then modify its version of the whiteboard appropriately (using a BlockingQueue to manage all of the concurrent edits users may be making) and relay these changes to each of the users separately. The users will then change their whiteboards to match the changes made to the server-side whiteboard.

Whiteboard:

Canvas canvas

Represents the model behind the actual whiteboard GUI.

String name

Represents the name of the whiteboard.

List<User> connectedUsers

Contains a list of users currently accessing the whiteboard. This will be used to show the identity of users concurrently editing.

void makeLineSegment(int x1, int y1, int x2, int y2, Color color, int thickness)

Draws a line segment on the canvas from (x1, y1) to (x2, y2).
This is intended to be called frequently, as freestyle strokes can be broken down into a

sequence of many line segments. The color and thickness will determine the stroke style.

String getName()

Returns the name of the whiteboard.

void setName()

Allows users to change the name of the whiteboard (given that the name isn't currently taken by another whiteboard).

List<User> getUsers()

Returns a list of Users currently editing the board.

boolean addUser(User user)

Adds a User to the list of users in the whiteboard. Returns true if the user is added without any problems; returns false if another user has the same username.

boolean removeUser(User user)

Removes a User from the list of users in the whiteboard.
Returns true if the user is removed without any problems; returns false if an error occurs.

User:

Whiteboard whiteboard

Represents the whiteboard that the user currently has access to.

String username

Represents the username that identifies the user on the whiteboard.

int thickness

Represents the current thickness of the strokes the user is making.

Color color

Represents the color of the strokes that the user is making.

void setColor(Color newColor)

Changes the color of the user pen.

void setThickness(int newThickness)

Changes the thickness of the user pen.

Color getColor()

Returns the current color of the user pen.

int getThickness()

Returns the current thickness of the user pen.

In Canvas, we will be making the following modifications:

drawLineSegment(int x1, int y1, int x2, int y2, Color color, int thickness)

Draws a line from point (x1, y1) to point (x2, y2) in the given color and thickness.

Protocol

Client-server interaction will use a simple HTTP protocol to transfer data. All interaction will be done using line segments drawn on the whiteboard. User-drawn strokes are broken down into small line segments, and the data about these line segments are sent to the server. The server, which has individual threads connected to each client, processes these line segments in order, adjusts its own version of the whiteboard, and then sends out the information to the other connected clients to make sure each user has an updated copy of the whiteboard. We use a central BlockingQueue that each of the separate client threads will use, so that we can guarantee that each client modification will happen in some order. A consequence is that if two strokes are

drawn over the same pixel at different times, the pixel's color will be that of the later stroke. Strokes that occur at the same time will be processed in an arbitrary order, so the color of the pixel will be that of the stroke that was processed last in this arbitrary decision.

The HTTP protocol that we will be using to transmit data will have the following syntax:

```
draw [INT] [INT] [INT] [INT] from [INT] [INT] to [INT] [INT]
```

The first three integers represent the color of the line (in RGB notation from 0-255), while the fourth integer is the thickness of the stroke. The first set of integers following "from" consists of the x and y coordinates, respectively, of the line segment origin and the second pair of integers consists of the coordinates of the line segment terminus. This protocol will allow simple communication between users and the server while providing all information needed to reproduce strokes on the whiteboard.

Concurrency

Here we are mostly using thread-confinement to ensure that we do not have deadlock or race conditions. We do not use many shared variables in our program, preferring to keep local versions and periodically updating them. For example, the whiteboards that each person sees is not shared, but simply maintained by constant message passing. We also ensure that there are no race conditions by putting a BlockingQueue in the central server thread, which will handle whiteboard modifications in some set order. While many client threads will be adding onto the BlockingQueue, because the BlockingQueue is a thread-safe data type, we will not run into any concurrency issues.

Testing Strategy

There will be two stages of testing – one for the view/controller and one for the model. The model will be tested using JUnit testing mechanisms – we can call individual methods and then check if the modified state is what we expected.

The abstract datatypes will have its various mutator and observer methods tested. Server and client interaction should also be tested using JUnit automated testing in order to make sure that the protocol is being parsed correctly and messages are being properly sent.

The GUI will be mostly tested by hand, but the testing strategy will be documented and included as an extra document.