Table of Contents

1. What we've done
2. What Doesn't work as expected
3. Setup

1. What We've Done

- **Server implementation:**

  UID: section/problem/extra
  Channels: /whiteboard/drawing/UID/ - handles passing of all whiteboard objects
  /whiteboard/chat/UID/ - handles chat messages

  The jetty server runs in **ANDYSCOMET**/cometd/trunk/ and listens to the publishes from the javascript files in /web-UI/andes/. Jetty, when started, runs the Cometd servlet. It first checks for any updates listed in the main pom.xml file. If there are updates, missing dependencies, or modifications to server code, maven will rebuild and pack. Jetty is then started on 127.0.0.1:8081.

  The Cometd servlet opens two channels, which are used to send information between the clients and the server. We chose to use two different channels to segregate the traffic. The first is drawing, which handles all object communication--the bulk of transmission traffic. A message is published to this channel whenever an object is modified (object-modified), created (new-object), selected (object-select), deselected (object-deselect) or deleted (delete-object).

  The second channel is chat, which as name implies, handles chat communication as well as helpful information from the server. Consult Chat for details.

  The class that we have modified inside the source code of the Cometd servlet is ServerChannelImpl.java. Every JSON object that is published by any client subscribed to the channel is handled by the publish method inside this class.

  If the JSON object is a chat message, the class ServerChannelImpl publishes the object as it is received. But if the JSON object corresponds to an object of the canvas, the class will parse the object and if necessary it will modify it (as part of the conflict resolution implementation).

- **Chat**
  - Once the Jetty server and the help server are up and running, the user can begin chatting. Chat is implemented inside the original hint side panel that was present in Andes. Every user is connected to the Comet server, and through this chat can be achieved. Upon a user typing into the hint box and pressing enter or the hint button, the chat text is converted to a JSON object which is then published to the chat channel, where all users who are currently subscribed to that chat

channel receive the JSON object and write it to the chat box html element. The JSON object has the pertinent information for such a device (poster's username, their message, etc.) and the client-side code interprets the object and prints its relevant information in chat format (username: text). There are also messages received to all subscribers upon a new subscriber joining the channel that state that the new user has entered the problem.

- File(s) Modified: help.js

- **Drawing**
    - Implementation of drawing was done leveraging the components of chat. We used a channel to handle all drawing updates. The channel is called *whiteboard*. There exist three different actions that can be done in drawing: (1) create a new object, (2) modify an object, and (3) delete an object. (There exist two other actions (object-select and object-deselect), but they are explained in the context of Conflict Resolution). These actions are published to each client as JSON objects. Examples of JSON objects for each of the actions are provided below::

1. create a new object:


```
{"id":"9","data":{"action":1,"user":"Joe","drawing":{"id":"andes.Combo0","text":"A. Square","x-statement":388,"height":177,"symbol":"","width":296,"action":"new-object","y-statement":76,"type":"rectangle","y":76,"mode":"unknown","x":82}},"channel":"/whiteboard/drawing/54321vec1ayQ_Joe"}
```


2.modify an object:


```
{"id":"13","data":{"action":3,"user":"Joe","drawing":{"id":"andes.Combo0","text":"A. Square","x-statement":388,"height":177,"symbol":"","width":296,"action":"modify-object","y-statement":76,"type":"rectangle","y":296,"mode":"unknown","x":496}},"channel":"/whiteboard/drawing/vec1ay54321Q_Joe"}
```


3. delete an object:


```
{"id":"16","data":{"action":1,"user":"Joe","drawing":{"id":"andes.Combo0","action":"delete-object"}},"channel":"/whiteboard/drawing/vec1ay54321Q_Joe"}
```


    - File(s) Modified: drawing.js

- **Conflict Resolution**
    - Objects can be created, selected, deselected, modified and deleted on client side. We followed the model of having ownership of objects. The first person that selects an object becomes the owner (or locker) of that object. On the server side we create a HashMap to keep track of who owns each object. Every time a client who is not the owner of an object selects said object, the client is notified that a conflict exists due to being 'owned' by another client.
    - The current implementation works as follows:
        i. Selection of an object: Each time a client selects an object that it does not own, it will receive a message from the server saying that there is a conflict. This message is sent by the server to all the clients connected to the channel, but since the client id is kept in the message from the server, each client will check the id from the message, and if the id from the message does not coincide with its id, it will ignore the message. If the

message id coincides with the id of the client, a message will be printed in the chat window saying that there is a conflict with that object, but nothing else will be done.

ii. Modification of an object: For any given object, let one or more clients have it object selected at the same time. Each of them can modify that object and deselect it. Let them follow any given sequence of modifying-deselecting the object. During that sequence of modifying-deselecting we will see two different behaviors in the screens of the clients, depending on whether a client has the object selected or not. But at the end of the sequence, every screen will show the same thing, and that same thing will be the modification done by the last person to deselect the object.

iii. Deletion of an object: Whenever a client that has an object selected deletes it, the object is deleted from every other client to (not mattering if they own it or not).

- File(s) Modified: drawing.js, help.js

- **Logging**
  - In the file Help/database.cl, we changed the logic for loading problems so that if the 'extra' field starts with "Q_" then it uses [section] AND [problem] AND [extra] to load previous work, otherwise it functions normally.
  - File(s) Modified: database.cl

2. **What Does Not Work as Expected**
   - **Chat**
     - The hint button works as the submit button for chat, this forces a hint on every chat publish sent, causing a compounding of hints while chatting.
   - **Conflict Resolution**
     - Delete
       - Let client A be the one who originally deleted the object. For any other client that is not A but that also has that object that was deleted by A selected, the object will keep showing on their screen, and it will stay there as a phantom object. Sometimes it will disappear, and other times it will stay.
       - Possible Fixes or Alternative Implementations :
         - Lock the objects on the client side as well, by marking them 'locked' (with a distinguished color) and not allowing a client to select any object that is already locked by a client. In order to achieve this, we would need to add a new field to the objects on client side with a locked flag.
         - Another alternative solution to the problem: Whenever a client selects an object, if the object has been locked by another client, the server will send back a message saying so. When the client receives this message, the client should

be forced to deselect this object. This solution does not require keeping track of any additional field or data structure on the client side.

3. **Set-Up**

The folder where our code is located from now on is called **ANDYSCOMET.**

1. Download the original Andy's code from: git://github.com/bvds/andes.git.
2. Download the Andy's Comet team code from: git://github.com/awaldow/AndysComet.git.
3. Copy the following files from **ANDYSCOMET**/changed_files/ to your code in /web-UI/andes/:
   a. drawing.js
   b. help.js
   c. menu.js

4. Copy the following files from **ANDYSCOMET**/changed_files/ to your code in /Help/:
   a. database.cl

5. Copy the following from **ANDYSCOMET**/dojox/ to /web-UI/dojox/:
   a. cometd.js
   b. cometd folder

6. Run the script andes_setup.sh. This script downloads source code that is needed for the Comet server to work.

7. Run startup.sh to start the Comet server.

startup.sh - This is a script that starts up the Comet server.
update.sh - This is an optional script that after any modification to server, will update dependencies, recompile code, and start server.