Indian Institue of Technology,
Kanpur

Topics in Distributed Systems(CS632A)

Course Project

Collaborative Whiteboard using RMI

Submitted by:
Group 16
Harshit Gupta(18111020)
Mohit Malhotra(18111042)

Instructor:
Prof. Ratan K Ghosh

November 10, 2018

Abstract

A Whiteboard is an electronic version of blackboard. Nowadays, it is widely used in various places for presenting ideas to a group of people but if multiple users want to collaborate to express their ideas, it can not be easily done with the simple whiteboard we know. Therefore, we have built a collaborative whiteboard system using Java RMI that would enable users to draw on their screens and get a combined view of all the screens.

1 Problem Statement

Many times multiple users may want to collaborate to present something, like multiple instructors collaborating in taking a class or some business meeting where multiple users may want to write or draw on the same screen. Doing this with a simple whiteboard is not a good idea as the users have to operate on the same screen one by one. This may even also not be practically feasible as the users can be at different places. To solve this issue, we have developed a collaborative version this whiteboard where each user can write on her screen and the same operation will be reflected on everyone else's screen as if they are not multiple but a single whiteboard that is being shared. This would enable users to contribute smoothly in presenting their ideas even if they are not close to each other.

2 Technologies used

1. The project has been implemented in java8. Reason for choosing java is its platform independence and robustness. Also, it is a object oriented language which would help in treating all the entities in the project without any hassle.

2. We learnt about java RMI during the course and were very impressed by its working model. Simple method calls for sharing of data makes it very useful for such applications.

3. Java Swing provides the support for building the GUI. Swing components follow the Model-View-Controller(MVC) paradigm, and thus can provide a flexible User Interface. Moreover, Swing components are lightweight, i.e., less resource intensive.

3 Implementation

3.1 Communication Model

We are using java Remote Method Invocation(RMI) for communication in our model. Stubs are generated for each user, this enable other users to execute methods(remotely) on them.

3.1.1 Registry Service

To start the application, we first start our registry service on a machine. The application starts **rmiregistry** on it's local machine and binds it's object with the name given as input so that the users can do a look up for it. This registry service performs the function of registering every user(collaborator) in the system and providing it remote objects or stubs of other users. The new user is also assigned a unique ID by the registry service which is updated by invoking a method on it's stub. As soon as a new user is registered, all other users are notified about this new user by providing them a remote object or stub for this new user.

The registry service maintains a list of current users and also an index table. When a new user registers, this list of users(stubs) is sent to that user. The index table(hash table) is used if a user wants to leave the system.

A user deletes itself from the system or network by sending it's unique ID to the registry service. The registry service looks up for this id in it's table to retrieve the corresponding stub. This stub is deleted from it's own list of users and then its id is sent to all other users to remove it. This is done by calling remote method for removing a user on the remote objects of all other current users which in turn deletes the user specified by the received unique ID.

3.1.2 Users

A user who wants to join the system(become a collaborator) must first get itself registered in the system. A user starts it's application by specifying a user name by which it would be known in the system, the name by which the registry service is bound in the **rmiregistry** and the address of the host on which the **rmiregistry** is running, i.e. machine on which registry service

is running. It also provides a color(in Hex code format). This color should be unique for this user as other users would differentiate this user's drawing on the whiteboard by it's color(a unique color for every user).

This user locates the **rmiregistry** by specified host address, then it looks up in the **rmiregistry** for the registry service to obtain its remote object. The user then invokes the remote method on this object to register itself by passing its remote object as a parameter. The registry service registers this user and returns the list of current users(their remote objects) in the system. This list would be used through out the execution to communicate with other users. The registry service also provides this user a unique ID which would be used for future operations. This user also maintains a index table(hash table) after getting the list of current users. It populates this table by using their IDs as index or key(received again by invoking remote method).

The user program has two methods(remote)- addUser() and removeUser() which are used to add and remove a user respectively. These functions are invoked by registry service, as already specified.

When a user leaves the system, the registry service is informed which then informs other users about its removal too. We have implemented this in such a manner that a user does not have to tell specifically that he wishes to leave instead he can just close the application and our program would automatically inform about its removal.

3.2 User Interface

Note: The UserInterface class is not for handling the UI part of the application. It is just the interface for the User class. The code for handling the UI is included in User class itself.

Our UI has a canvas for the user to draw. On the top right corner, there is a list of collaborators in the system. Below it are buttons for handling different operations as described below.

1. **Write:** Clicking this button will enable the free drawing mode.
2. **Erase:** To erase the drawn stuff.

3. **Line:** For drawing a straight line. The first click would specify the starting point, then the collaborator can hold and drag the pointer. A line would be drawn from the starting point to the releasing point.

4. **Rectangle:** Drawn similarly as line. A rectangle will be drawn with line from starting point to releasing point as diagonal.

5. **Oval:** Same as rectangle. An oval will be drawn that fits the corresponding rectangle.

6. **Clear:** This button is used to clear the white board(on all screens).

7. **Save:** This button stores the current state of the whiteboard as an image.

Drawing geometrical shapes(line, rectangle and oval) was not as easy as it seems to be because there is no direct support for that in java swing package. We had to take an instance of current image of canvas when the drawing of shapes started and for every dragging and release operation, we just refreshed the screen with that image and the shape with these new dimensions.

The color of the drawing depends upon the collaborator who had drawn it.

3.3 Data Sharing

For free hand drawing operations, a user can draw on his screen and the updates would be displayed on his screen immediately but not transferred to other users until the user releases the mouse click or lifts the pen. As soon as it does, the list of points drawn is transferred to all other users currently in the network and they draw it on their screens too after receiving it. Similar actions take place for erase operation also.

For geometrical shapes, only the starting point(where the mouse was pressed) and the releasing point is transmitted as soon as the user completes drawing the shape.

With all these operations, a tag is also passed which specifies what is the operation to be performed(write, erase, line, etc.). In addition, the sender's color is also passed as a parameter to the remote method call which draws

on the receiver's screen. These calls are made on all the remote objects in the system.

For clear operation, only a tag is necessary. When the method is called on the remote object, it clears the whiteboard on that corresponding user's screen.

To maintain consistency, a user can either draw and send or receive updates at one time. If one operation is being performed, the corresponding updates for the operation are stored in a buffer which are displayed on the screen when the current operation completes.

Batches are used in updates because it would reduce transmission messages and thus network traffic.

For the collaborators who join late, they will first check if there were some collaborators already in the system and if there were, it would invoke a remote method on one of the remote objects of a user already in the system requesting it an instance of the current state of the whiteboard. This method will return an image which will be drawn on the user's whiteboard as soon as it starts the application so that he can catch up. It doesn't really matter which remote object's is invoked as they are in the same state, the model being consistent. So, without any loss of generality, we will chose the remote object at the head of the list.

Transmitting this image is not also directly feasible as in RMI, the transmitted data must be serializable and images are not. So, we had to first convert an image into an array of bytes and then transmit this array of bytes which we will reconstruct to an image at the invoker.

We also have included a save option that would save the current state of the whiteboard as a png image.

4 Some notable features

Although we have explained almost everything in the implementation section, some key notable features of our implementation that make this project a

useful tool are:

1. Apart from simple drawing, we have provided a few geometrical shapes that would reduce the work for the user of this application and as well make the drawings more neat and attractive.

2. The state of the whiteboard can also be saved as an image which can be used for different purposes in different applications like minutes of the meeting, lecture notes, etc.

3. The list of all the current collaborators is present which enables a user to know about other collaborators in the system.

4. If a collaborator joins or leaves the system, it does not affect other users.

5. With multiple registry services, even on the same machines we can create multiple group of collaborators in the same network.

6. By sending very few updates and that too of very small size(just the points to be updated and not the whole whiteboard), our system is fast and does not lead to a lot of network traffic.

5 Future works

1. Some other features could be added to the white board like inserting images, filling colors, arcs, text, animations, etc.

2. Batch sizes can be varied according to the feedback from users.

3. This application can also be modified to work as a tool that can run over other applications. This can be very useful where we want to draw over presentations, web pages, videos, images, etc.

4. To add even more features to this application, we can embed voice conferencing and/or text(transcripts) transmission.

6 Conclusion

We have successfully applied a middle ware tool(RMI) that we learned about in this course to implement a shared version of whiteboard. A lot of challenges were faced and resolved.. that increased our skills in developing networking applications. We have tested our application on different machines and different platforms and it seems to perform well. We tried it on with ten collaborators(on 3 machines) and it performed reasonably well. The transmission was fast enough.