Distributed Collaboration - Assignment 2: Traceable N-User IM/Editor

Date Assigned: Saturday September 13, 2014

Submission Date: Sep 22, 2014(11:55 pm)

Objectives:

- Understand how to implement traceable/testable single-site and distributed algorithms
- Understand how to use application-level multicast
- Understand how to created replicated models

In this assignment, you will first make your 1-User version of the IM/Editor project traceable. Next you will extend it create an N-user IM/Editor. You will use the GroupMessages (library) multicast layer to implement your application. Your code will use the replicated architecture where each user process will have a local model that will be kept consistent with local models of other processes. You will trace two additional steps in your N-user implementation. As extra credit, you will communicate status changes among the users and trace these changes.

You will need to pull the latest changes from ColabTeaching for this project. Before each assignment, be sure to pull any changes that have been made.

All assignments will most probably be changed after they are posted to clarify them or change constraints, and features. The changes will be highlighted using track changes.

Tracing 1-User IM

To provide better documentation and also support automatic grading, trace your 1-user tool of the previous assignment. Use the four Trace objects mentioned in class: ListEditInput, ListtEditMade, ListEditNotified, and ListEditObserved. You should fire these events for both lists: the topic, which is a list of characters, and the history, which is a list of strings. These events take as arguments the names of the lists. You should use ApplicationTags.IM and ApplicationTags.EDITOR as names of history and topic lists, respectively. ListEditInput and ListEditMade should be announced after the change is input and made to the list, respectively, ListEditNotified should be announced just before the notification method is invoked, and ListEditObserved should be announced at the start of the notification method. ListEditNotified is announced before the actual operation that notifies observers~~cation~~ so that it precedes ListEditObserved in the log.

Multi-User Replicated IM

Implement the multi-user IM tool along the lines shown in the second demo and illustrated in the figure below. Both the console and GUI views should be updated in response to remote actions – here and in ~~the~~ my demo posted on YouTubue, only the GUI is shown. In your demo, you should show both views change

You will create at each site the architecture you implemented in the previous assignment (a local model and two interactors). This architecture will be extended to ensure that the local models are kept consistent, ignoring issues of causality and concurrent input. This means:

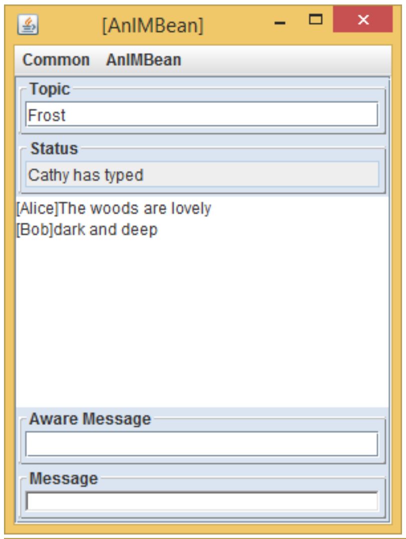
1. Each history contains messages entered by each user in the collaborative session. The name of the user should prefix the message in the history entry using the syntax shown in the figure below. As in the last assignment, a message string is entered in a history when the user hits ENTER.
2. The topic contains characters entered by different users (without an indication of which user entered each character). Each character should be multicast as soon as it is input, the ENTER key has no meaning as in the previous assignment. This is why the topic had to be modelled as a list of characters and not as a String. (I mentioned this fact in class; in case you did not follow that, please look at the edits I made to the last assignment detailing this issue.)
3. If you implemented the extra credit status object, as extra credit for this assignment, you should now maintain in the model a table of (key, value) pairs where the key is the name of the user and the value is the current status. The interactor will now display a status string constructed from the table that shows the status of each user. You are free to choose the syntax of the display. In fact, you can choose the widget you use to display the table – for instance you can display it in a Swing table.

The identities, locations and other properties of the collaborating users should not be known before the collaboration. Moreover, any user should be able to join the IM. You can assume that no user enters the session after the first message is input; that is you do not have to support latecomers. You will not use RMI or some other IPC mechanism directly. You should use the GroupMessages library described in class, which trivially supports these features. You are free to use relayed or direct communication. The PPT slides describe both the concepts and the small API needed in this assignment. In addition, I have linked a copy of my conference paper on it from the chapter column of the web site.

Tracing N-User IM

You should now fire two additional traceable events, ListEditSent and ListEditReceived, just before your code multicasts topic/history changes and just after it receives these changes, respectively. ListEditSent is announced before the actual multicast so that it precedes all the traceable steps taken by the multicast library to send the list edit.

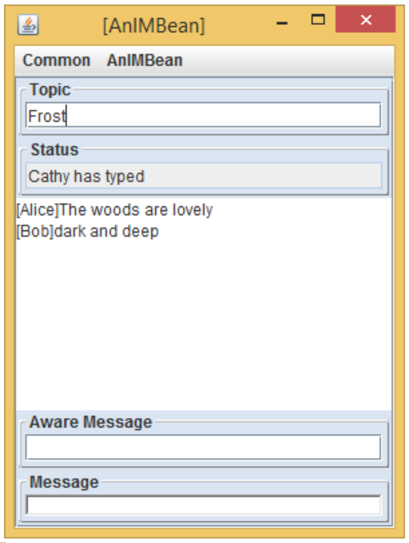
If you implement the extra credit feature, you should fire four new events, TableChangeMade, TableChangeNotified, TableChangeSent, and TableChangeReceived, respectively. The times when they should be announced should follow the logic of the list edits. As name of the table, you should use ApplicationTags.IM.



Frost

Cathy has typed

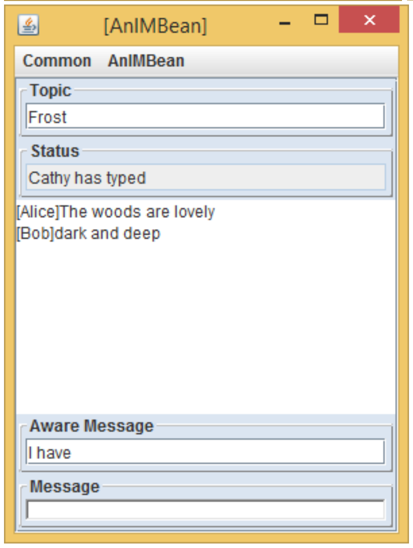
[Alice]The woods are lovely
[Bob]dark and deep



Frost

Cathy has typed

[Alice]The woods are lovely
[Bob]dark and deep



Frost

Cathy has typed

[Alice]The woods are lovely
[Bob]dark and deep

I have

Make sure the tracing in on in your clients for the events you fire. To do so, you should execute the following code before you display your user interface:

```
Tracer.showInfo(true);
IMTraceSetter.traceIM();
```

Formatted: No Spacing

Main Class and Tags

In the previous assignment, you provided (at least) one main class (class with main method) to run the echoer. In this program you will provide and run at least N+ 1 main classes, one for the session manager, and one for each of the N clients in a session. As in the code and demo shown in lectures, you should support at least 3 clients in the session. As in the lecture code, the session manager main class should have the tags: *DistributedTags.SESSION_MANAGER, ApplicationTags.IM* and the main class for client^k should have the tags: DistributedTags.*CLIENT_k, ApplicationTags.IM*. Both kinds of classes can have additional tags such as *DistributedTags.SERVER* for the session manager.

Formatted: Normal

Extra Credit Demo

The extra credit requires you to support three states. Please each transition possible between each pair of states. Thus, show the transition from typing to has entered and has entered to typing. Similarly show the transition from no action to typing.

Formatted: Normal

Submission

By the submission date, submit a link to a video (with an audio narration) showing your IM tool in action and also submit your code to Sakai. Please post the link as a private ~~or public~~ message in Piazza rather than an email (which is hard to keep track of) or a Sakai submission (the overhead of logging in to Sakai is high). Tag the link as hw2videolink. If you are worried about privacy issues, free to use the other means, and post a Piazza message informing me you have done so