Distributed Collaboration - Assignment 3: Replicated N-User IM

Date Assigned: Sep 24, 2009

1-user IM Completion Date: Oct 1, 2009

N-user IM Completion Date: Thu Oct 15, 2009

Deleted: 8

Objectives:

- Implement a collaborative application.
- Implement replicated, distribution - unaware model.
- Use and evaluate distributed pair and side-by-side programming.
- Understand how IM is implemented.
- Understand Remote Method Invocation.
- Understand properties and other programming patterns.
- Understand state-change notifications associated with various patterns.

In this assignment you will implement an N-user instant messaging tool. We will make some simplifying assumptions about the UI and the collaboration features to make your task easy. Initially you will implement a 1-user IM and later extend it to N-users using remote method invocation. You will implement only the model of the application – the user-interface will be provided by ObjectEditor. Each user will interact with a separate instance of the (root) model class. The states of the models of different users will be kept consistent to implement N-user semantics. However, the models themselves should be distribution- unaware though they can be collaboration-aware. All distribution awareness should be added to separate modules, which can themselves be centralized and/or replicated. Try to minimize the amount of collaboration-awareness in the models.

As in the previous assignments, work in pairs, and try and choose someone with whom you have not worked so far.

As this is a programming assignment, this is an opportunity for you to experiment with two recent distributed collaborative programming practices: distributed pair and side-by-side programming. Distributed side-by-side programming requires you to have access to two computers – let me know if that is an issue. As before, you will record your collaboration sessions using LiveMeeting.

The assignment is broken into multiple parts to allow you to incrementally build it. To make sure you do not procrastinate – there are two completion dates.

The assignment involves the use of Java RMI and the Java-based ObjectEditor. Moreover, later, your models will be incorporated into two other Java-based infrastructures. Therefore you need to program in Java.

Simplifying assumptions

You can make the following simplifying assumptions:

- *Single session*: There is only one IM session supported by your tool. Thus, the name of the session does not have to be specified.
- *Text and editing-status coupling*: There is no information shown/ communicated beyond the text messages, and who has entered but not committed a message. For instance, the time of a message does not have to be displayed or communicated.
- *Personal information*: The only information about a participant kept by the tool is his or her name – thus, you don't have to worry about associating the person with a picture or status.
- *UI generation*: The UIs are generated by ObjectEditor. The library is available from the Web page. To use it, simply include it in your class path.

You are free to ignore the assumptions. However, if you decide to implement your own user-interface, you should use MVC with the programming patterns discussed in class. To make sure you in fact do so, first create the application using ObjectEditor, which will force you to meet this constraint. You can later replace ObjectEditor with a custom UI implementation.

1-User IM

Create a 1-user IM that takes as a command line argument the name of the user invoking it. The IM should allow a user to enter an input message and view the history of all entered messages in the order they were entered. As in a commercial IM, each entry in the history should be associated with the name of the user. Finally, if the user has entered but not committed an input message, the IM should display this information. For extra credit, you can try, as IM tools do, to distinguish between input fields that (a) are being actively edited, (b) have been edited but later cleared, and (c) have text that is not being actively edited.

To implement this application, sketch out the logical structure of the information displayed to the user, and create one or more models that follow programming patterns exporting this structure to the view. These patterns should be the ones discussed in class. As mentioned above, use ObjectEditor to create the view and controller of the models. You don't have to worry about the nature of the user-interface. In fact, your logical structure should be entirely independent of the user-interface. To determine if you have followed this rule, you should try to imagine multiple user interfaces for the IM and make sure your logical structure supports all of these UIs. If you do not follow this rule, you have not created models.

You are free to use the ObjectEditor predefined classes, util.AListenableString, util.AListenableVector, and util.AListenableHashtable, to implement the models. In case you need to extend the functionality of these classes, you can use inheritance or delegation, that is, create classes that have IS-A or HAS-A links to these predefined classes. Inheritance will be much easier to use.

Distributed Pair Programming

Your implementation of the 1-user IM should involve at least 90 minutes of distributed pair programming. In such programming, two developers work synchronously from different computers, using a telephone to talk to each other. One person, called the driver, inputs code, while the other, called the navigator, views the code. They continuously discuss the code. After 45 minutes, the developers switch navigator and driver roles. Use LiveMeeting to create a pair programming session by sharing the desktop of one of the developers.

If the response of the person at the slave computer gets intolerable, switch seats (assuming you are nearby) or create a new LM session wherein the master and slave computers are switched. Record your session. Another alternative is to use a two display side-by-side setup, with users taking turns using their primary computers (see below).

If you have 3-people in your project, do a 135 minute session, and make each person a driver for 45 minutes.

N-User IM

Extend the 1-user IM to support an arbitrary number of users. As in the previous version, each user starts a program that takes his or her name as an argument. All users who run the program are put into the single IM session. Do not worry about leaving the IM session – that is, implement Hotel California semantics not allowing people to leave! Each user should be able to identify all users who have entered but not committed their input fields.

All users in the session should see the same set of messages, though the order of messages in their histories can be different if these messages were sent concurrently.

As mentioned before, the models displayed by ObjectEditor should reside on the local computer. This means and you should implement centralized and/or replicated modules that synchronize the local models. As mentioned earlier, the models should be distribution-unaware and minimally collaboration aware; and the remaining components can be centralized and/or replicated.

Use Java RMI for distributed communication.

Distributed Side-by-Side Programming

Your implementation of the N-user IM should involve at least 60 minutes of distributed side-by-side programming. As in pair programming, in such programming, two developers work synchronously from different computers, using a telephone to talk to each other. However, they can work concurrently, and use some shared/replicated store to share their changes. In addition, each person can see the desktop of the other user on a special computer dedicated to providing this awareness information. Thus each user interacts with a primary computer and an awareness computer and shares the desktop of his primary computer, which is shown on the partner's awareness computer.

Distributed pair programming is a special case of distributed side-by-side programming, wherein only one primary computer is active at one time. In this two-display set-up, response times are not an issue because each developer uses the local computer. Unlike pair programming, side-by-side programming does not impose a process – you are free to determine how much concurrent and pair programming you do.

Again use LiveMeeting and record both desktop sharing sessions. If you have 3 people in your project, either use three computers or allow only two to do concurrent work.

Side-by-side programming requires you to share your code changes. You are free to determine how you would do this. If you use a version control system, simply use check-in and check-out to share them. If you have access to a shared file system, through AFS, NFS, or shared windows folders, then you can use it. You are always free to create a directory in my deposit directory to do so. The most interesting way to share these changes is through Microsoft Groove (which comes with Vista Office and is available for free trial for about a month on XP and other platforms). Groove can instantly synchronize private folders on multiple computers.

Written Document

Write a document that:

- Describes how you divided your work when you were not doing pair programming, that is, when you were doing traditional asynchronous programming and synchronous side-side programming.
- Compare these three forms of programming, giving their pros and cons.
- Explain the kind of discussion that was useful when you were doing synchronous pair and side-by-side programming.
- Write a document describing the architecture of your implementation. Be sure to identify how much distribution and collaboration awareness there is in the various components of your implementation. Classify components according to the function they provide such as model and session management rather than the specific classes.
- Trace what happens when three users enter messages at the same time. In fact, also test this condition for at least two users. You might have to disable firewalls to do so. I

think when you start your rmi server, it will ask you to block or unblock the progra, which should set the firewall property.

- Has test cases, in the form of screen figures, describing the functionality you implemented. Be sure to explicitly indicate the extra credit features you implemented.

Submission

Submit a printout of the document.

Put your code and documents in a subdirectory in the folder: \\dewan-cs.cs.unc.edu\deposit\790-063-Assignments\Assignment3

User Interface

The nature of the user-interface is not important in this assignment, but if you want to improve it, I will try and help you.

The general way to improve the UI is to set UI attributes of some property, P, of class C. These attributes must be set before you ask ObjectEditor to edit an object.

Here are generalizations of some questions I have had so far:

1. *How do I display C.P in a text area widget?* You can display any property, P, of class C, in a text area widget. By default it will display the value returned by invoking the toString() method of the value of the property. You can also ask it to format this String. (You will probably need to change the default size of the widget and attach a scrollbar, which are explained below.)Here is what you have to do:
   // ask ObjectEditor to format C.P instead of using the toString() value
   ObjectEditor.setPropertyAttribute(C.class, "P",
   AttributeNames.UNPARSE_AS_TO_STRING, false);
   // use JTextArea to display C.P
   ObjectEditor.setPreferredWidget(C.class, "P", javax.swing.JTextArea.class);
2. *How do I influence the size of the widget used to display C.P:*
   // set the width and height of C.P
   ObjectEditor.setPropertyAttribute(C.class, "P", AttributeNames.COMPONENT_WIDTH, width);
   ObjectEditor.setPropertyAttribute(C.class, "P", AttributeNames.COMPONENT_HEIGHT, height);
3. *How do I attach a scrollbar to the widget used to display C.P:*
   // attach a scrollbar to C.P, which will show only when needed
   ObjectEditor.setPropertyAttribute((C.class, "P", AttributeNames.SCROLLED, true);

Deleted: ¶
¶