Distributed Collaboration - Assignment 2.3: Model, Interactor, and Interactor Generation and Coupling

Date Assigned: September 20, 2012

Completion Date: October 5, 2012

Objectives:

- Understand the decomposition of an application into model and interactor.
- Use interactor generation.
- Couple generated and manually created user interfaces of different users along with the telepointers

Model and Interactor

Redo the single-user version of your application by creating a model and interactor object for it. You are free to separate the interactor into a view and controller. You should follow the Bean conventions for models and observers, that is, implement getters and setters and Bean property notifications. In addition, you should use ObjectEditor conventions (or predefined classes such as AListenableString) for lists, tables, shapes, and notifying strings. We should cover these on Sep 24th. The notifying strings will be essential also for allowing multiple users to concurrently edit different parts of the IM topic text. In the case of a notifying string, you will not use a setter to receive changes to it. Instead you will listen to vector events fired by it. However, if you want the string to be editable in ObjectEditror, you have to currently define a setter for it.

Interactor Generation

Use ObjectEditor to automatically generate an additional user interface for your model. Thus, each user will have two interfaces for interacting with the application. The ObjectEditor library (oeall21) is available from the download section of the course page

Model-based Telepointer

Make your telepointer shape also a model that is displayed by a glass pane interactor. Follow the ObjectEditor conventions to define this shape. We should cover these on Sep 24th.

Using your Telepointer Model in ObjectEditor

Call the setGlassPaneModel() method in an OEFrame returned by ObjectEditor.edit() with your telepointer shape model. This method returns a JComponent glasspane

This glasspane draws the shape but does not redispatch it to the components of the JFrame it covers.If you have written a universal redispatcher abstraction, use it to send the events in this glass pane to the components of the JFrame in the ObjectEditor user interface. This JFrame is returned by the getFrame() method of the OEFrame.

If you have not written a redispatcher, then instead of calling the setGlassPaneModel() method, call the setTelePointerModel() method, which uses the redisptach method I have written.

Coupling the User Interfaces

Ensure that changes made in a user-interface result in corresponding changes to all corresponding local and remote user interfaces, including changes to the tele-pointer. This means your interactor and glasspane must be distribution unaware and all shared state must be in the IM and telepointer model. Of course, the telepointer will not point to the same widgets in the ObjectEditor and manual user interfaces, but it should be at the same location. You should show actions in an ObjectEditor user-interface reflected as changes in remote ObjectEditor and manual interfaces, and changes in a manual user-interface reflected as changes in the remote interfaces. Be sure to consider all kinds of actions including telepointer movements, new message entry, status change, and editing of shared text.

Submission

Create and post a video showing the coupled user interfaces, which in turn, should show all the new features you have implemented. In addition, give me a printed document giving

1. the Java interfaces of all event processor and telepointer abstraction and how the telepointer abstraction has been used in the two applications of it (telepointer and non telepointer)
2. all the regular credit features you have implemented
3. all features you have implemented that go beyond what was asked. These could include the extra credit features I suggested and additional features you chose.