IBM

developerWorks®

UML Activity Diagrams: Detailing User Interface Navigation

Benjamin Lieberman , Ph.D. April 29, 2004

from The Rational Edge: The author illustrates how to use UML Activity Diagrams to capture and communicate the details of user interface navigation and functionality, and explain three stereotypes: presentation, exception, and connector.

This is the third and final article in my series for The Rational Edge *on using Unified Modeling Language (UML) Activity Diagrams. In the first and second articles, I showed how Activity Diagrams could be used effectively to capture use-case flows of execution and also to detail the dynamic aspects of the Process View.*[1] *In both cases, this involves altering and enhancing the basic semantics of new stereotypes for activities. These stereotypes better capture details of the problem domain for use-case execution flows and the dynamic nature of system processing for the Process View. In this article, which illustrates how to use UML Activity Diagrams to capture and communicate the details of user interface navigation and functionality, I reintroduce three of these stereotypes: presentation, exception, and connector. The focus is on how to apply them to the functionality available to system users via a graphical user interface.*

Most modern software is designed to interact, at one time or another, with sapient, bipedal life forms (aka, users). To facilitate communication between computer systems and these creatures, software developers invented graphical user interfaces (GUIs). These interfaces are often based on some form of "windowed" environment populated with *presentation* elements that represent the current system state, *controls* for altering the current system state, and *navigation* to shift the focus of attention from one section of the system to another.

The visual elements may be windows, dialog boxes (modal and modeless), frames, bars, blocks and panes, etc. Controls take the form of input boxes, sliders, dials, buttons, selection lists, grids, check boxes, radio buttons, etc. Navigation, made even more important with the advent of Web-based displays, takes the form of directional buttons (forward, back, sideways), menus, links, hotspots, hyperlinks, and more. This bewildering combination of display, control, and navigation offers the potential for an extremely rich and powerful interface, but it also makes for a distinct challenge to the system modeler, who is expected to communicate not only the visual aspects of the system, but also the exact behavior expected from each of these elements.

Capturing Visual, Control, and Navigation Elements

Activity Diagram Icons and Stereotypes

In constructing Activity Diagrams, it is helpful to use colored icons (and UML stereotypes) to indicate specific activities and visually differentiate various steps in a flow.(See Figure 1.) This is particularly important for connector icons (pointers to additional diagrams) that link to separate use-case scenarios.*

Action is the primary diagram element. This icon represents activities performed by the System or Actor. Since it is the most common icon, it typically has either a neutral color or no color at all.

Presentation activities are indicated by the <<presentation>> stereotype. This stereotype indicates that there is a conversation between the use-case Actor and the System. It represents a special category of *Action* activities and is used to abstract user interface details.

he *Connector* stereotype represents connections to flows diagrammed elsewhere. The use of Activity Diagrams often leads to the creation of large and complex models, so it is useful to indicate alternate navigational flows. The connector icons for this stereotype can be used to automatically link to another diagram (e.g., to a separate Rational Rose diagram using an embedded link). If desired, the extension to a Rose (UML) diagram can lead to a "subactivity" diagram embedded within the activity itself. One caution, however: This approach can rapidly produce a very "deep" model with multiple, embedded layers. Such a model runs contrary to the ideal of a high-level "roadmap," which shows an overview and leaves the details for the textual description.

Exception activity occurs when there is an exceptional flow in the use case, and is indicated by the <<exception>> stereotype. This usually represents an error condition but may also represent unusual or unexpected system behavior. If the exception is an error condition, then it is useful to summarize the error inside the icon. The icon is also useful to indicate system logging and recovery activities.

The *Frame* and *Page* stereotypes are useful in describing both Web pages and application displays. *Frame* is used to represent areas of the presentation that are logically or physically separated. *Page* is used to indicate separate display pages on a Web site. A *Frame* is often logically associated with a *Page*, but this is not strictly necessary. For example, a *Frame* may be used for universal site navigation and as such is not linked to any specific Web *Page*.

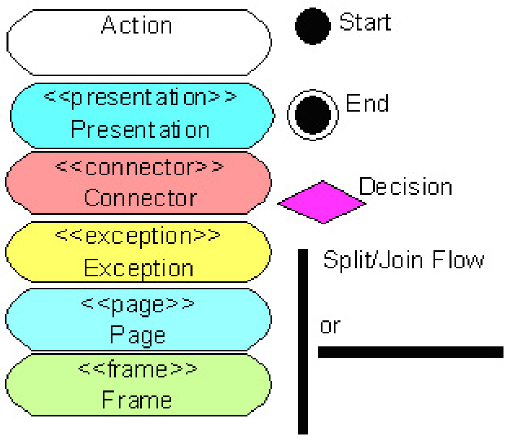
It would be easy to expand our list of Activity Diagram stereotypes and icons, but this represents a fairly complete and simple set for modeling user interfaces. Since the purpose of these diagrams is to enhance understanding of complex flows, adding new icons (and stereotypes) should be done with caution.

For most system modelers, the set of tools available to accomplish this complex task is somewhat limited, consisting mostly of flowchart semantics and screen shots of the interface. Although flowcharts are effective for capturing the dynamic aspects of an interface, they lack the cohesive capability inherent in UML ? to translate and connect concepts from one model to another (e.g., sequence diagrams and class diagrams). Moreover, static screen shots cannot illustrate the sequence of events that occur during a dynamic user "conversation" with the system. Screen shots show *what* you can do, but not *how* you can do it, and they cannot illustrate the effect on system state regarding a specific user action.

UML Activity Diagrams are well suited to capturing a series of actions taken on behalf of a user via a GUI. Although on the surface Activity Diagrams may seem little more than glorified flowcharts (and perhaps not even as powerful because they lack specific symbology), in fact they represent the same highly adaptable, extensible, and specific modeling structure evidenced in the remainder of the UML specification. With the addition of several well-chosen stereotypes (part of UML's extension mechanism, along with *constraints* and *tagged values*), Activity Diagrams represent a strong addition to the user interface modeler's tool kit. As will be illustrated in detail below, Activity Diagrams provide the ability to accurately depict the dynamic aspect of virtually all current user interfaces.

Figure 1 shows an overview of Activity Diagram elements and icons. See the Sidebar for a description of the Activity Diagram icons and stereotypes we will use in our model.

Figure 1: Overview of Activity Diagram Elements and Stereotypes



I am not the first person to suggest adapting UML semantics for this purpose. David Anderson, for example, published a paper in which he refers to a Navigation Model to describe the user experience with the visual elements of the system.[2] He sets forth a number of desirable traits for a given model, including that it be easily drawn, implementable by a CASE tool, allow for expression of navigation from one element to another, indicate parallel activities, and express user or system actions. I believe that UML Activity Diagrams represent a well-developed technique for satisfying many of these requirements. Although Anderson does use UML statecharts effectively to represent navigation from one page or system state to another, I would argue that Activity Diagrams show these changes in a more natural and easier-to-visualize manner.

Scott Ambler also recently authored a book chapter that recommends the creation of Interface-Flow Diagrams to illustrate the movement from one "screen" to another; the transition between one system state and another is indicated as the result of a user action (e.g., "click search button," or "select customer")[3]. As we will see below, this approach can be very neatly applied using the guard condition semantics and activity elements from Activity Diagrams.

The Graphical User Interface

As I explained in my two previous articles for *The Rational Edge*, Activity Diagrams give you the ability to construct a "roadmap" of user functionality that accurately represents the paths a user *can* follow, but they do not discuss *why* or *when* a particular path should be followed. This information is best captured in the use-case textual description, which allows you to express the required level of detail. Similarly, process flows are paths that the system can take, but the detail on when and why is found only in the Software Architecture Document artifact of the Rational Unified Process®,[4] or in similar system description documentation.

This article presents a technique for modeling three common but distinct types of GUIs: a Web site; a guided interface (e.g., "wizard"); and a standard application interface. For each type of interface, the details of the user experience ? including branching, selection, navigation, and constraints (enabled/disabled menus or controls) ? will be captured in the model.

As with all modeling techniques, the task begins by segmenting the modeling domain into sections that can be organized and managed. For this purpose, we will use the elements we identified above: **presentation** (manages appearance), **controls** (manage state), and **navigation** (manages location).

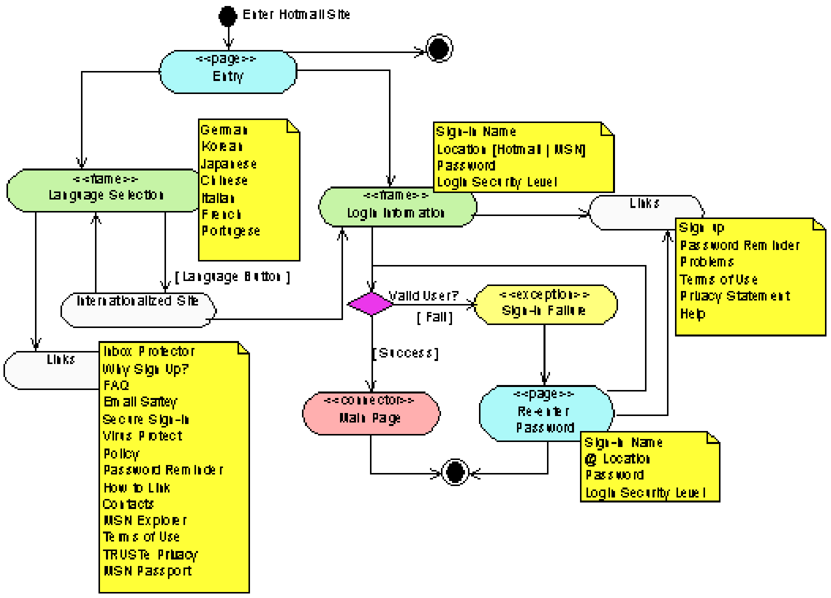
Modeling a Sample Web Site Interface

As an example of a Web site interface, we will model the Microsoft? Hotmail® Web site (www.hotmail.com).[5] As shown in Figure 2, the starting point for the Web site is the Entry page, which consists of two frames: a Language Selection frame and a Login Information frame. The term *frame* indicates that the functionality is logically separated, but not that it is necessarily implemented using HTML-based frame tags (although it may very well be, in which case the designation is exact). As shown in the figure, users can select a different display language (all of which are indicated), and the system will then present them with the same interface branded to the selected language.

Users may also select from a collection of hypertext links that will take them to different pages or sites (note: to save space, I will not include a full description of these sites in the example; a document should accompany the diagrams to indicate the transfer location ? whether it is internal or external to the modeled site).

The user login information consists of the Sign-In name, the location (MSN.com or Hotmail.com), the password, and the security setting (high, low, or no security, with links to explain the various levels). If the user is validated, he or she will then see the initial Hotmail screen. If the user is invalid or nonexistent, a "Sign-In Failure" page will appear that asks the user to reenter the information. Note that although the information is nearly identical in the login frame and the re-enter page, they are shown as separate because the user interface distinguishes between them.

Figure 2: Hotmail Login Site

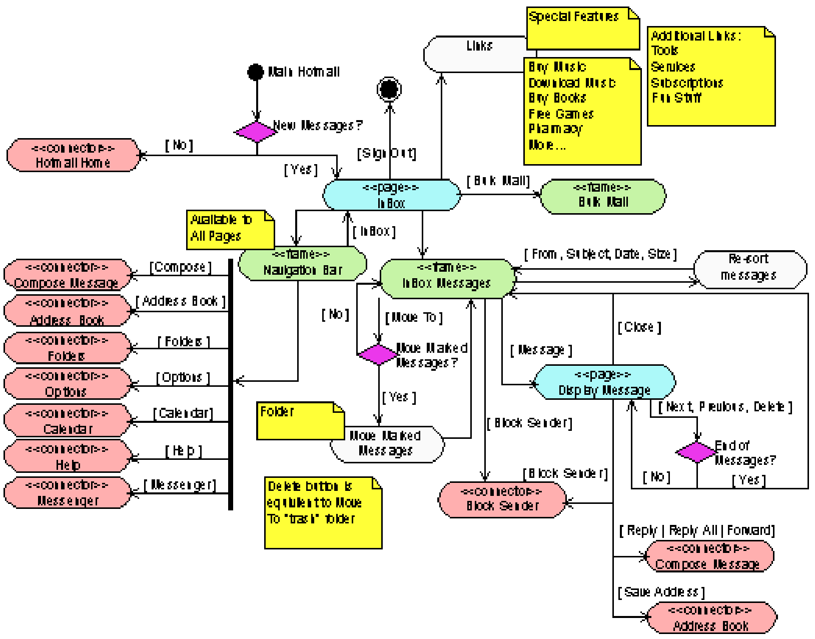


After successfully completing the login, the user is taken to the main page for the Hotmail system (Figure 3). From here, the user is presented with either the Hotmail homepage if no new messages have been received, or the user's InBox if new messages have arrived. The InBox offers numerous options for navigation and state control. The navigation bar, which offers options for composing messages, the address book, calendar, etc. , is available from all pages within the site. The InBox frame allows the user to change the state of the display and sort messages according to From, Subject, Date, and Size. Messages may be marked for deletion, blocking, or movement to other designated folders.

By selecting a message, the user can bring up the Display Message page and access additional navigation options for his messages, such as Next, Previous, and Delete. If the user clicks on Reply, Reply All, or Forward, he is taken to the Compose Message page; the requested addresses are already displayed in the message header (shown here as a <<connector>> activity). Finally, the user may chose to add the sender to his Address Book or to block further messages from the sender.

In Figure 3, notice the use of frames to segment functionality and the placement of links into notes. Often a Web site makes many hyperlinks available to users (advertisements, off-site links, etc.). Usually, it is only necessary to diagram those links that will take users to another section of the current site (Navigation) or change the state of the display (Controls). The guiding principle here should be to "tell a story" with your diagrams. Too much information may confuse the reader and make the model less useful. Frames are a good way to segment the functionality presented by complex sites (such as the Hotmail site). Separating concerns visually (i.e., Navigation and Control) will facilitate modeling of the site. Note, however, that <<connector>> rather than <<frame>> is used for the off-page links. This is done deliberately to show that these links lead to other diagrams or some other change of user focus. The <<frame>> stereotype, as used in this example, indicates the current focus of user attention: the InBox and associated messages.

Figure 3: Hotmail Inbox and Message Display



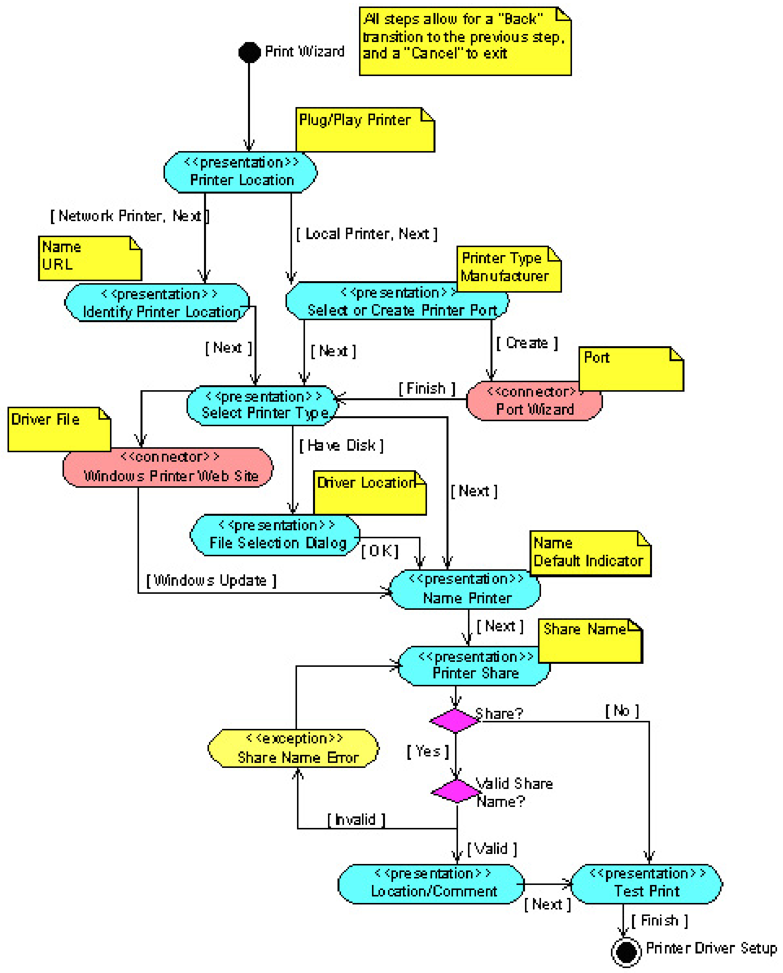
As an aside, an interesting extension to this approach might be to use a new generic stereotype called <<navigate>> to indicate a toolbar, drop-down, or other navigation-based page element that had not been fully specified (during site construction).

Modeling a Guided Interface

To illustrate the modeling of guided interfaces (e.g., "wizards") we will use the Print Wizard in Windows 2000,? which assists users in creating new printer ports and installing print driver files. As shown in Figure 4, the Print Wizard is mostly a series of dialogs that lead the user from one step to the next. The functional flow is as follows: First identify the printer type that will be used (networked vs. local); then select the printer port (or network location), the printer driver, and printer identifier; finally, generate a test print page. As Figure 4 shows, there are several branches in the flow (e.g., the creation of a new port, shown by the Port Wizard <<connector>> icon) as well as data elements that are either optional or mandatory at each step. If the user does not provide a mandatory data element (such as Share Name), then an exception will occur and an error message will be displayed. When the user follows all the steps correctly, then the printer connection will be established and the Print Wizard will end.

In this model, the note at the beginning indicates that the user has the option at any step to return to the previous step (back) or to cancel the operation (cancel). Also, there is only one exit point from this wizard, although some guided interfaces provide the option to end the sequence early (such as the Internet Connection Wizard). In this case, the guard [finish] condition may occur in earlier steps and lead to a different exit state. Also, this series of steps allows the user at one point to go online to select a driver update from the Microsoft? printer Web site; this is indicated by a <<connector>> activity and may or may not be modeled elsewhere. In any case, the resulting information (here the updated driver files) is indicated. As would be expected from a directed interface, navigation is limited, and there are very few explicit backward connections (we already noted the ability to back up one step at a time) or other looping connections.

Figure 4: Print Wizard User Interface Flow

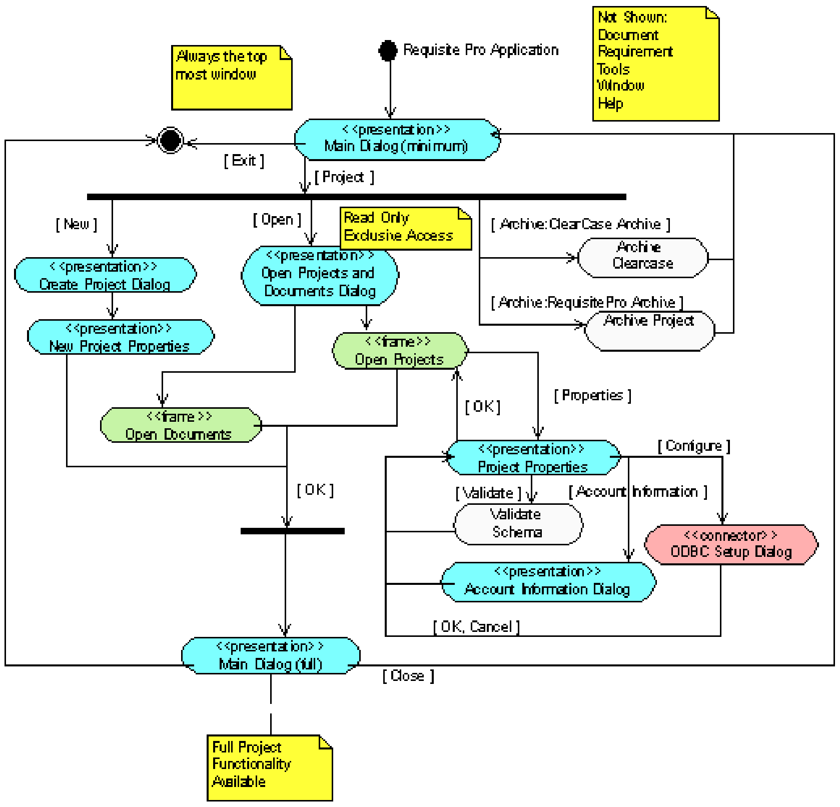

Print Wizard
All steps allow for a "Back" transition to the previous step, and a "Cancel" to exit
Plug/Play Printer
<<presentation>> Printer Location
[Network Printer, Next]
[Local Printer, Next]
Name URL
<<presentation>> Identify Printer Location
Printer Type Manufacturer
<<presentation>> Select or Create Printer Port
[Next]
[Next]
[Create]
Port
[Finish]
<<connector>> Port Wizard
<<presentation>> Select Printer Type
Driver File
<<connector>> Windows Printer Web Site
[Have Disk]
Driver Location
[Next]
<<presentation>> File Selection Dialog
[OK]
Name Default Indicator
[Windows Update]
<<presentation>> Name Printer
[Next]
Share Name
<<presentation>> Printer Share
Share?
[No]
<<exception>> Share Name Error
[Yes]
Valid Share Name?
[Invalid]
[Valid]
<<presentation>> Location/Comment
<<presentation>> Test Print
[Next]
[Finish]
Printer Driver Setup

Modeling an Application Interface

Our final example will be to study one aspect of a standard application and capture the functionality and navigation available to the user in a model. For this example we will explore the Rational RequisitePro® user interface. The RequisitePro application opens with a display of the main navigation menu (which is always the top-most window). Figure 5 shows the functionality available from the Main Dialog project menu. The other menu choices (not shown) can be captured using the same mechanism. As shown in the figure, there are five possible selections from the project menu when no project is available: New, Open, Archive (ClearCase and RequisitePro), and Exit. By selecting New, the user opens the "Create Project" dialog. Selecting Open brings up the "Open Projects and Documents" dialog and offers two frames: one for projects, one for documents. With the projects frame, the user can open the properties dialog that will allow validation of the project, setup of the ODBC database, and presentation of account information. Once a project is open, then the Main Dialog is altered to allow for full functionality.

Not shown in this example is the security check dialog, or the remainder of the program functionality. The remaining system functionality, however, can be modeled in much the same way I described above. Buttons, dials, sliders, switches, and other controls can be shown by the changes they produce in system state. Dialogs, windows, frames, boxes, and other presentation mechanisms can be illustrated in the same manner as the initial project set-up. Finally, navigation around the system via menu selections, buttons, or other navigation mechanisms can be captured as transitions between screens and system states.

Figure 5: Requisite Pro Main Menu Functionality


Always the top most window
Requisite Pro Application
Not Shown: Document Requirement Tools Window Help
<<presentation>> Main Dialog (minimum)
[Exit]
[Project]
[New]
[Open]
Read Only Exclusive Access
[Archive:ClearCase Archive]
Archive Clearcase
<<presentation>> Create Project Dialog
<<presentation>> Open Projects and Documents Dialog
[Archive:Requisite Pro Archive]
Archive Project
<<presentation>> New Project Properties
<<frame>> Open Projects
[OK]
[Properties]
<<frame>> Open Documents
<<presentation>> Project Properties
[Configure]
[Validate]
[Account Information]
Validate Schema
<<connector>> ODBC Setup Dialog
[OK]
<<presentation>> Account Information Dialog
[OK, Cancel]
<<presentation>> Main Dialog (full)
[Close]
Full Project Functionality Available

Activity Diagrams: A Powerful Tool for System Architects

This article has presented a technique for capturing the navigation of various user interfaces, including Web sites, assisted functionality (e.g., wizards), and standard applications. In each case, Activity Diagrams can illustrate possible navigation paths through the interface and connections to other parts of the system functionality. This technique provides a clean way to model and represent possible paths through the interface and key data elements required for state changes. For organizational purposes, it is useful to segment the system functionality by presentation, controls, and navigation elements of the interface.

For each type of interface, presentation elements are the windows, dialogs, and other visual elements of the interface. Controls are shown by their effect on the system (e.g., sorted messages), and navigation is represented as transitions between activities.

Like the other two articles in this series, this one illustrates the power and value of Activity Diagrams for communicating system functionality, processing, and user interface flows. For the system architect, these Diagrams expand the tools available for the expression and modeling of complex software systems.

Notes

[1] As explained in Philippe Kruchten, "The 4+1 View Model of Architecture." *IEEE Software*, 12 (6), November 1995, pp. 42-50.

[2] David J. Anderson, "Extending UML for UI." SprintPCS.com, Kansas City, MO (2000), http://math.uma.pt/tupis00/submissions/anderson/anderson.html

[3] Scott W. Ambler, "User Interface Design: Tips and Techniques" in *The Object Primer: Building Object Applications That Work, and Process Patterns*, 2e. Cambridge University Press, 2001.

[4] Philippe Kruchten, *The Rational Unified Process: An Introduction,* 2e. Addison- Wesley, 2000.

[5] Hotmail is a registered trademark of Microsoft, Inc. The functionality modeled here is subject to change, as Microsoft alters its Web site; it should not be taken as a representation of all available services.

*****For a Rational Rose®script to automate the application of color to activity model elements, see the Appendix below.

Appendix: Rational Rose Activity Diagram "Colorizer" Script

This Rational Rose® script (for version 2000 and greater) will automatically add diagram element fill color to the Activity Views on each Activity Diagram included in the use-case model (e.g., under the Logical View root package).