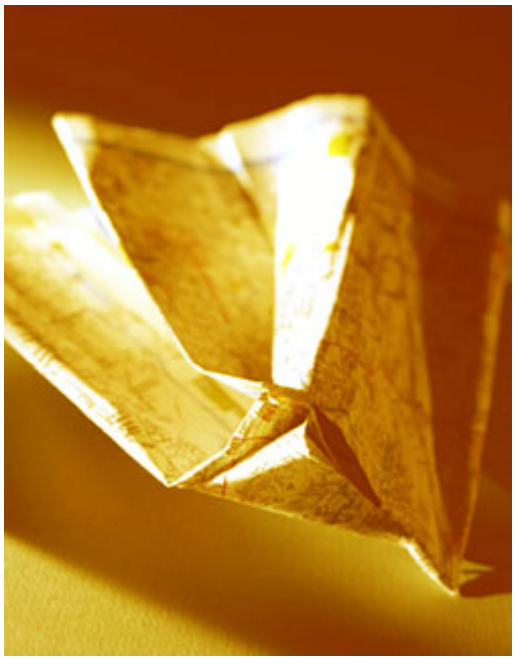
 



# UML Activity Diagrams: Versatile Roadmaps for Understanding System Behavior

by **Ben Lieberman**
Senior Software Architect
Blueprint Technologies



*The core purpose of software development is to provide solutions to customers' real problems. Use cases[1] are a vital aspect of a technique that has been used successfully to ensure that development projects actually focus on these problems. They are used to discover, capture, and present customer requirements in a form that is accessible to developers, testers, and other stakeholders in a development project. To detail a use case, it is critical to capture basic, alternate, and exceptional flows of execution, which represent major and minor threads of execution the system encounters as it processes customer requests.*

*Using the "standard" use-case form,[2] these flows can be captured using plain English to describe sequential activities (see Figure 1). These descriptions are quite detailed, however, and they can be difficult to decipher -- especially within a complex set of use-case scenarios.*

*This article describes another way to capture these flows: by using Unified Modeling Language (UML) Activity Diagrams that depict the flows as "roadmaps" of system functional behavior. These roadmaps are analogous to AAA (Automobile Association of America) roadmaps, in that they show what routes you can take but do not indicate whether you will take them. An AAA map, moreover, supplies only enough information to identify locations of interest, leaving detailed descriptions of the road for companion travel guides. Similarly, Activity Diagrams show a comprehensive summary of use-case flows but leave the design details up to other artifacts.*

*We will also take a brief look at other ways to use Activity Diagrams during the development lifecycle.*

**Basic Flow:**

1. The User requests login to the system.
2. The User enters login ID and password.
3. The System validates the User's permissions.
4. The User is presented initial system menu choices.
5. [...the use case continues...]

**Alternate Flow:**

1. In Step 2 the User requests a new password.
2. The User enters the login ID, and new and old passwords.
3. The System validates the User's Permissions and continues at Basic Flow Step 4.

**Exceptional Flow:**

1. In Step 3 of the Basic Flow and Step 2 of the Alternate Flow the User enters either an invalid login ID or an incorrect password.
2. The System returns an error condition with the string "The User login ID and/or password is incorrect."
3. Processing resumes at Step 2 of the Basic Flow.

**Figure 1: Textual Descriptions of Basic, Alternate, and Exceptional Use-Case Flows**

## Activity Diagram Overview

The primary consumers of Activity Diagrams are the customer stakeholder, testing team, and software development staff. For them, these diagrams form the visual "blueprints" for the functionality of the system, as described and detailed in the use cases. Tracing paths (threads of execution) through these Activity Diagrams enables all stakeholders in the process to understand and monitor the development of system functionality.

The latest Unified Modeling Language (UML) specification, version 1.3,[3] describes Activity Diagrams[4] as a mechanism to capture business workflows, processing actions, and use-case flows of execution. Although the Rational Unified Process (RUP®) uses Activity Diagrams to detail

activities for each of the nine workflows recommended for software development (Figure 2), it offers few other examples of Activity Diagram applications.

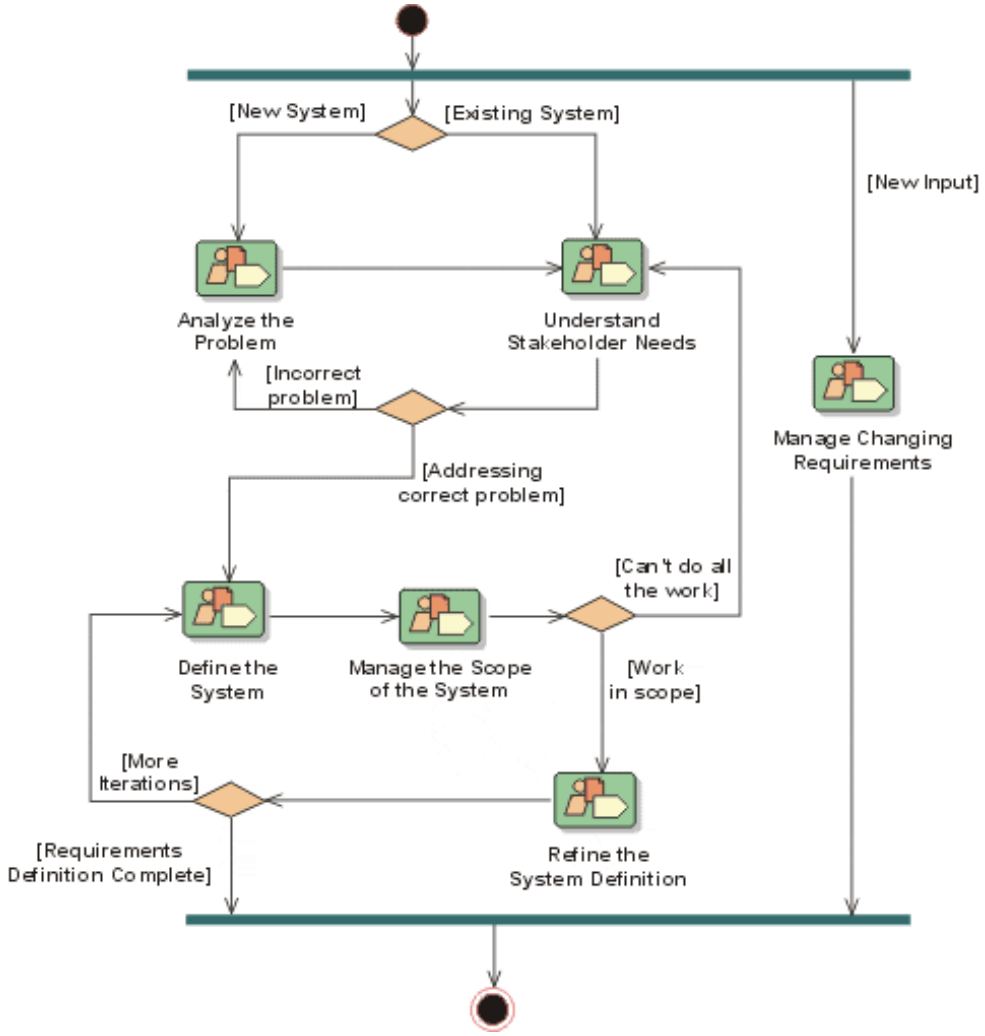



**Figure 2: A Rational Unified Process Activity Diagram Illustrating the Requirements Workflow**

In fact, Activity Diagrams can be used for many purposes: diagramming use-case flows; modeling complex business operations or processes (such as the one in Figure 2); depicting data and information flows; and even computing algorithms.[5] In addition, as we will discuss, they can be used later in the development lifecycle for system impact analyses and to develop and track test cases. For a brief introduction to the standard icons and stereotypes used in Activity Diagrams, see the Sidebar below.

## Example Use Case: Maintain User Profile

To understand the practical utility of Activity Diagrams for mapping use-case flows, let's walk through a realistic example of a use case for maintaining a user profile within a travel reservations system. To gather the

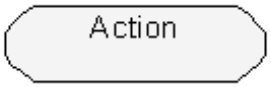
information needed for this use case, typically the System Analyst would conduct interviews with the subject matter experts to gain an understanding of the problem domain.[7] The analyst could actually capture the information from these interviews directly in an Activity Diagram, or she could first write up a textual description of her findings and then create an Activity Diagram to illustrate it.

In our example, we will use Rational Rose to illustrate the development of an Activity Diagram based on a use case for maintaining an information profile for a specific customer. The use case establishes the initial boundary points for entry and exit; each step in the use-case flow will be shown as a set of activities and activity flows.

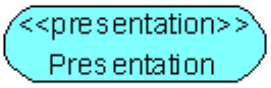
Figure 3 shows two activities from a simple use-case: 1) The user modifies his customer profile (a Presentation activity); 2) The system updates the information to a persistent store (shown here as a database icon). Note that there is no need to show all the processing steps at this stage; a typical session takes a top-down approach, starting broad and then narrowing the focus. Additionally, rather than representing the

**Activity Diagram Icons and Stereotypes**

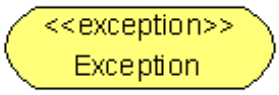
In constructing Activity Diagrams, it is helpful to use colored icons (and UML stereotypes) to indicate specific activities and visually differentiate various steps in a flow. This is particularly important for Off-Page icons (pointers to additional diagrams) that link to separate use-case scenarios.[6]



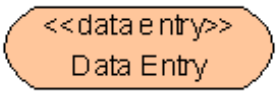

*Action* is the primary diagram element. This icon represents activities performed by the System or Actor. Since it is the most common icon, it typically has either a neutral color or no color at all.




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.




*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 (see Figures 4 through 7). The icon is also useful to indicate system logging and recovery activities.



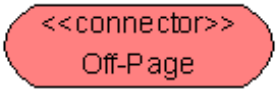

*Data Entry* activity is indicated by the <<data entry>> stereotype, which represents significant Actor interaction with the System for the purpose of adding, modifying, or removing data. Data Entry activities can range from simple field editing to complex visual rendering changes. This icon should be used with care to avoid cluttering the visual model with low-level data manipulation details. See "Set a Level of Abstraction" below for suggestions.

concept of persistence as a separate icon, it can be embodied right in the activity name (e.g., Save to Persistent Store). Persistence is a very important part of almost every system, and it is often beneficial to show explicitly where it occurs. This information is of particular value to the Test team who need to determine where and when in a test case the information in the Persistent Store needs to be verified.

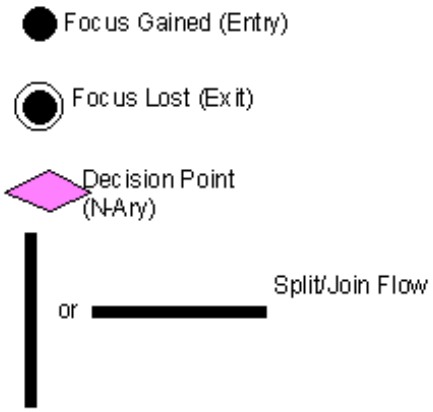
Arrows are used to indicate transitions from one action to another. The guard conditions on the transitions from the User Modifies Profile action indicate the possible paths presented to the Actor, shown here as [OK] and [Cancel].

## Paths Must Have Entry and Exit Points

Since we are modeling process flow, we must include a path through the functionality that allows the user to enter and then exit the functional area to move to another. If such a path does not exist, then there is a very serious error in the model (and possibly in the system itself).




The <<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 flow or extension points in use-case scenarios (see Figures 4 through 7). The Off-Page 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 (Unified Modeling Language) 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 "road-map," which shows an overview and leaves the details for the textual description of the use-case. Although you can use this icon to indicate <<extends>> and <<includes>> use-case relationships, often these are best represented in the main use case diagram. If they are depicted on the Activity Diagram, then they should be shown as coming off Decision Points (diamonds) with guard conditions to <<connector>> activities.




*These are additional icons.*

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 use-case activity flows. Since the purpose of these diagrams is to enhance understanding of complex use-case flows, adding new icons (and stereotypes) should be done with caution.

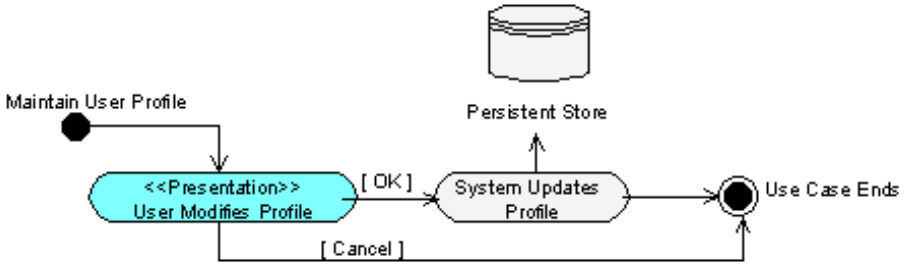



**Figure 3: Initial Activity Diagram for the Maintain User Profile Use Case**

Now, let's consider the same use case again, with the additional requirement that security must be in place for the viewing of sensitive information. Figure 4 shows the resulting diagram.

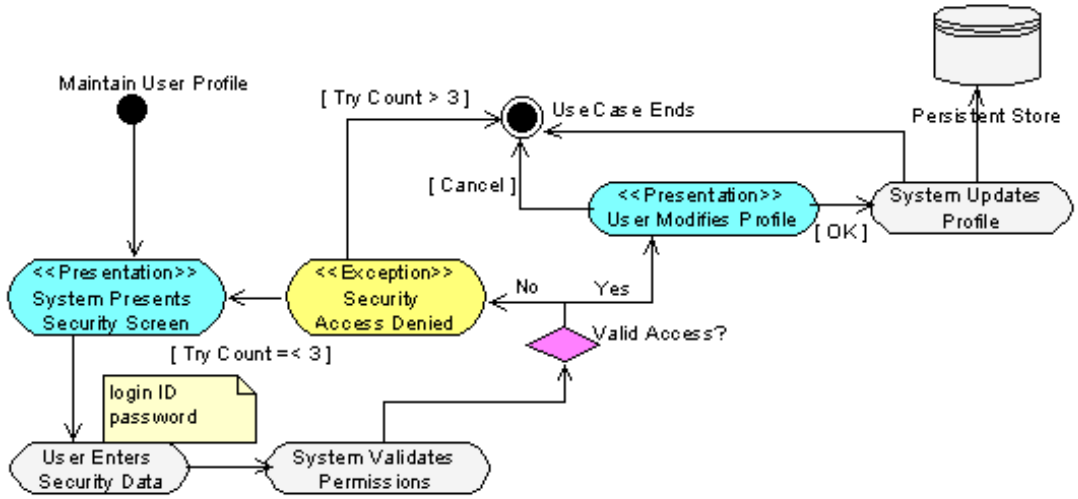



**Figure 4: Addition of Security Flows to the Maintain User Profile Use Case**

## Leave the Details for Other Artifacts

Note that we have added two more icons to the diagram to represent a Decision Point and an Exception, respectively. Also note that the Decision Point asks if the user has valid access privileges but does not detail the permission criteria. That level of detail will be found in the use-case textual description. In addition, the login ID and password data elements are indicated next to the User Enters Security Data activity. Data elements important to the use-case flow should be indicated in a note as shown, with the remaining data left to the use-case text for full elaboration.

Finally, the diagram indicates that the Security Access Denied Exception will return to the System Presents Security Screen Presentation activity until the Try Count exceeds three attempts; then the use case will end. Note that the Exception is declarative, but the specific actions (e.g., display of an error dialog box or message) at this juncture are detailed in the Exceptional Flow section of the use-case document and Graphical User Interface (GUI)

design screen shots (if they exist).

## Set a Level of Abstraction

Next, let's explore some additional processing requirements. We will assume that the user needs to change his name and address, and that the system needs to assign him a customer priority category (e.g., VIP, Senior Citizen, Employee, etc.). The diagram now appears as shown in Figure 5. We have added some Data Entry activities to indicate the user's ability to change certain data elements. This may not represent the complete set of editable data elements, but it does include elements important to the processing flow. Note that the Data Entry begins and ends within the Presentation activity. This implies that the user may repeat these actions as often as necessary. This approach is intuitive for system users, who expect that the system will return to a "wait" state after they perform an action.

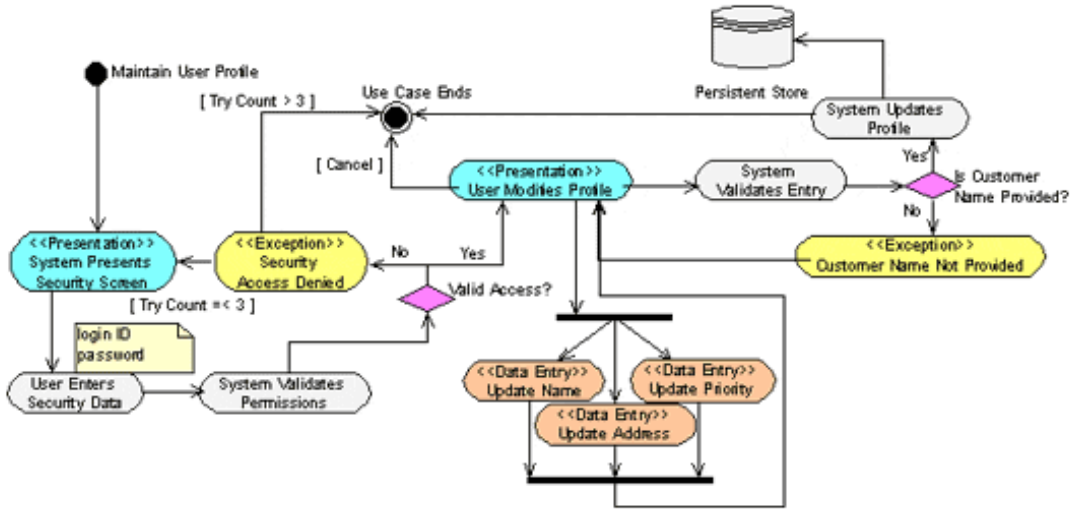



**Figure 5: Addition of Data Modification Flows and Validation Steps to the Maintain User Profile Use Case**
**Click here to view full size image.**

Now let's add more complexity to the model: We will assume a mandatory field for the customer name that must be correctly filled in before the user can exit the use case, as shown in Figure 6. We note the Name field next to the Presentation activity to show that it is important to the use-case flow. We have added a new Exception for when the customer name is not specified, and indicated that the Exception will reenter the main flow at the User Modifies Profile activity.

Finally, we will indicate that the User Modifies Profile activity can modify information about the user's travel preferences (assuming that this is part of the customer profile). We add the Off-Page (<<Connector>>) activity to indicate a link to another use case or use-case scenario. The name of the Off-Page activity should match the name of the use case or scenario referred to.

By now, the diagram has grown quite complex, so we can re-factor to further abstract activities. For example, we can collect all of the Data Entry activities into one activity or split portions of the diagram to separate illustrations and then connect them with an Off-Page activity. In addition, some of the activities (such as System Validates Entry) may be further elaborated in a separate diagram that we indicate by simply applying the Off-Page icon (<<connector>> stereotype).

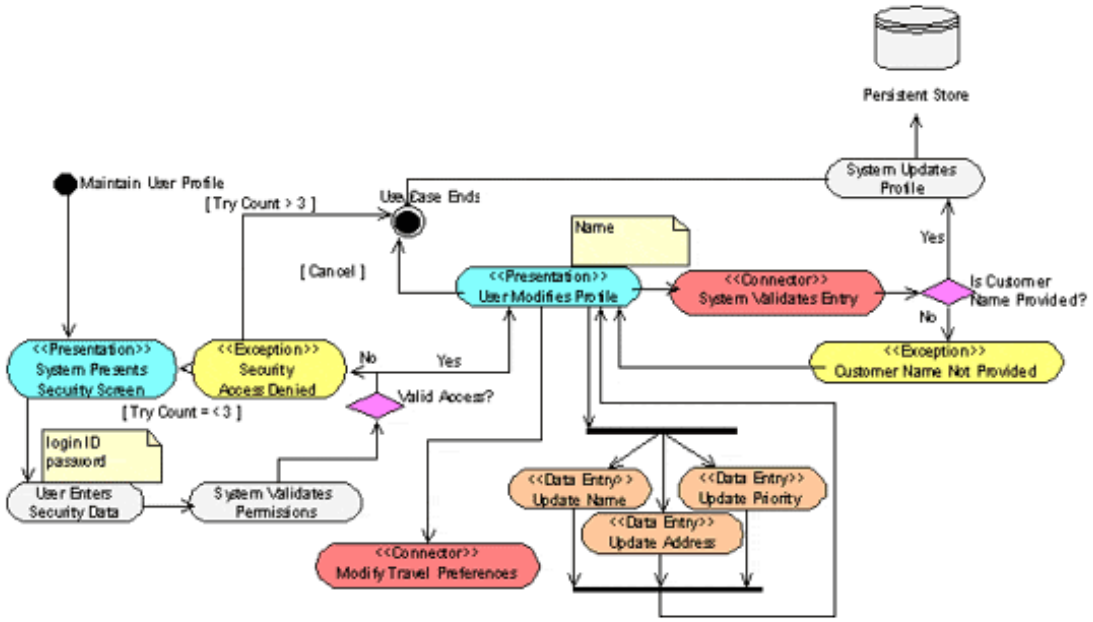



**Figure 6: Inclusion of Travel Information and Use Case Connectivity in the Maintain User Profile Use Case**
**Click here to view full size image.**

As a final example of this type of diagram, Figure 7 shows how UML Swimlanes can be used effectively to show interactions among the various actors and the system. This is not vital (the previous diagrams do not show this, for example), but it can increase understanding of which participant in the use case is responsible for which activities.

The example in Figure 7 is a credit card payment submission. The use case begins with a Presentation to the customer that specifies the credit card payment; the customer then enters and submits her card details. The system validates these values and either returns to the customer if there is an error or submits the payment to the Credit Card Service. If the card payment is accepted, then the system notifies the customer of success. If not, then the error is logged, and the customer is notified of the failure (and perhaps directed to handle the payment some other way). Note that it is easy to add features such as error handling if the Credit Card Service is unavailable, and also additional system accounting activities.

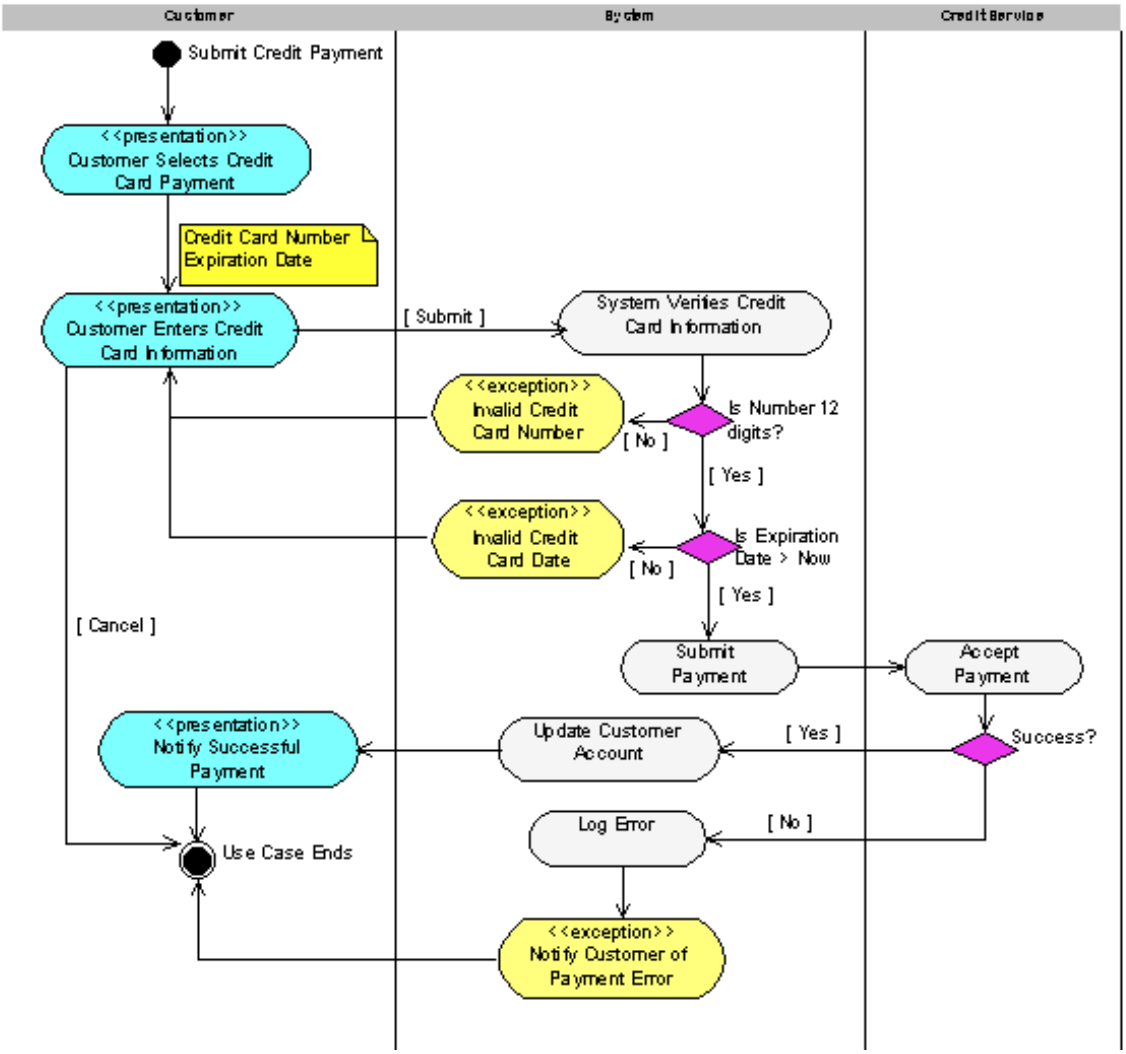



**Figure 7. Use Of Swimlanes in an Activity Diagram to Indicate Actor/System Boundaries and Responsibilities**
**Click here to view full size image.**

## Using Activity Diagrams: Freedoms and Restrictions

As is evident from the variety of uses discussed above, Activity Diagrams allow for a great deal of freedom. They encourage the creator to use the right level of detail to "tell a story" about the system functionality. A model is a communication device, after all, so it requires an adequate level of detail to address the problem to be solved. Clarity and brevity are important to avoid visual overload, but a model should present key features of the use-case flows.

In creating Activity Diagrams, you should also observe a few key guidelines:

- **Don't attempt to show system design elements.** A common mistake when doing use-case specification is to move into the solution space before adequately defining the customer's true needs.

A core principle of use-case specification is to focus on functionality the customer desires. If you create activities such as "Send Update Command to Profile Manager" or "Obtain Oracle Database Connection," then you are violating this key principle. The use-case Activity Diagram should serve as a guide to further analysis and design, not as a repository for design information.

- **Don't substitute activity diagrams for use-case descriptions.** The use-case flow diagrams are intended to *summarize* and *supplement* textual descriptions in the use cases, not replace them.
- **Limit the level of complexity for each diagram action.** As we saw in the example of the Maintain User Profile Use Case (Figures 4-7), the addition of more than three Data Entry activities should be collected into a common activity or split off into a separate diagram (as for the Travel Preference information in Figure 6). Use the following rules of thumb to limit complexity:
    - If there are more than three possible paths (alternate or exceptional), then use additional Activity Diagrams to promote understanding.
    - Use additional Activity Diagrams if the processing requires specific data elements.
    - Use Swimlanes to separate concerns, particularly for Actor/System interfaces. See Figure 7 and the related discussion under "Set a Level of Abstraction" above.
    - Do not use Activity Diagrams in this context to capture detailed system processing. Under no circumstances should low-level design information appear on these diagrams.
    - Display as much of a use case as possible in a single diagram. If you are constrained by printable page size, then consider purchasing a large-carriage printer rather than forcing a complex diagram to fit 8.5 x 11 inch paper. Alternatively, make use of the Off-Page icon (<<connector>> stereotype) to logically separate models.
    - Use a tool to maintain consistency for your models. Currently, no tool will automatically update Activity Diagrams linked to use cases, but most tools (e.g., Rational Rose 2001) will allow you to embed a diagram into the use-case model.
- **Maintain your models.** To have maximum benefit, your Activity Diagrams must be updated when use cases are modified. You can ensure this will happen by inserting the diagrams directly into the use cases as appendices. Moreover, the diagrams should be maintained in the same repository as the use cases. Rational Rose allows Activity Diagrams to be collected under a particular use case and for the textual representation of that use case to be linked to the same model location. This facilitates the update process and enhances the likelihood that the models will not become outdated.

## More Uses for Activity Diagrams

We have looked carefully at how to use Activity Diagrams for mapping flows of execution through a use case, but there are other applications for them as well during the development lifecycle. These are explained briefly below.

**System impact analysis.** During system maintenance and enhancement, the development staff receives many requests to locate and repair system "issues" or faults, as well as add new functionality. Use-case Activity Diagrams can be used to access the likely functional impact these changes will have on the system. By tracking Activity Diagram flows into the analysis and design models (e.g., by tracing to object sequence diagrams), you can quickly identify modules and subsystems that will be affected by proposed system changes. The changes can then be reflected in the activity models by changing the outlines of the activities to a different color (e.g., red) or thickness. This allows the test and architecture teams to rapidly assess what testing resources are necessary as well as the level of potential system breakage.

**Test case development.** Test cases are derived from use cases.[8] Therefore, use-case Activity Diagrams can be used to create specific scenarios for each test case. This can be done by tracing a thread of execution from entry to exit through each diagram, one for each test scenario. Activity Flow Diagrams are an excellent means for the test designer to scope the test for expected system behavior.

**Test case coverage tracking.** If the test team is not using automated methods to track use-case test coverage, then they can use Activity Diagrams to show the progress of a testing effort. They can designate paths as major and minor to indicate testing priority. They can also highlight the diagrams to indicate which activities they covered with each test. In this way, the Activity Diagrams can provide a visual representation of test progress for each functional area of the system.

## Overall: A Highly Useful Design Artifact

The UML is an excellent design and architecture language that has become the *de facto* standard for software system description. As we have seen, UML Activity Diagrams are particularly well suited for the discovery and visualization of complex functional process flows based on system use cases. Displaying these flows visually greatly improves the level of communication and understanding between the development staff and the customer. In addition, the test team can use these diagrams to directly aid in the creation of the test plan and test cases. Overall, Activity Diagrams represent a useful addition to the collection of design artifacts available to the software engineer.

## Appendix

Rational Rose Activity Diagram "Colorizer" Script

This Rational Rose script (for Version 2000e and higher) will automatically add fill colors to the icons for Activity Views on each Activity Diagram included in a use-case model (Use Case View root package).

**Want more information and advice on creating better use-case descriptions? See "Managing Use-Case Details" in this issue of The Rational Edge.**

---

[1] See Ivar Jacobson, Magnus Christerson, et al., *Object-Oriented Software Engineering: A Use-Case Driven Approach*. Harlow, Essex, England: Addison-Wesley, 1992. See also Dean Leffingwell and Don Widrig *Managing Software Requirements*, A Unified Approach. Boston: Addison-Wesley, 2000.

[2] See Ivar Jacobson, Magnus Christerson, et al., *Object-Oriented Software Engineering: A Use-Case Driven Approach*. Harlow, Essex, England: Addison-Wesley, 1992. See also Dean Leffingwell and Don Widrig, *Managing Software Requirements, A Unified Approach*. Boston: Addison-Wesley, 2000.

[3] See http://www.rational.com/uml/index.jsp for detailed information.

[4] See Grady Booch, James Rumbaugh, et al., *The Unified Modeling Language, User Guide*. Reading, MA: Addison-Wesley, 1999. Also see James RA: Addison-Wesley, 1999.

[5] See Grady Booch, James Rumbaugh, et al., *The Unified Modeling Language, User Guide*. Reading, MA: Addison-Wesley, 1999.

[6] For a Rational Rose script to automate the application of color to activity model elements, see the "Appendix".

[7] See Dean Leffingwell and Don Widrig *Managing Software Requirements, A Unified Approach*. Boston: Addison-Wesley, 2000.

[8] The Rational Unified Process, 2000.

---

***For more information on the products or services discussed in this article, please click here and follow the instructions provided. Thank you!***