Getting Started With Activity Modeling

An Oracle White Paper
May 2007

ORACLE

INTRODUCTION

Activity modeling is about creating a diagram that represents the dynamic aspects of a system by showing the flow of control from one action to another, and is one of the main UML modeling techniques. This paper discusses how activity diagrams can be used and how to create them using JDeveloper 10.1.3, and Oracle Consulting's best practices with that.

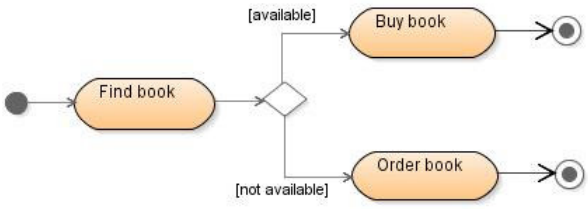
WHAT IS ACTIVITY MODELING?

As has been stated in the introduction an activity diagram represents the dynamic aspects of a system. Its purpose is to focus on flows driven by internal processing, rather then by external events.

Activity diagramming is used to model the dynamic aspects of a system. It is best known for detailing use cases, but can be used for other purposes as well.

Activity diagrams perhaps are best known as a way to detail use cases. In particular an activity diagram can be used to illustrate the scenario of a use case (see also the white paper "Getting Started With Use Case Modeling"). But its usage is not restricted to use cases, as in principle the level of modeling can vary from business-level activities (like the life cycle of a bank account) to a piece of program code (like the one responsible for updating the balance of an account).

The scope of the diagram is a task with a clear starting point and that finishes by reaching some end point with a clear defined result. The task is executed by performing one or more **actions**. The diagram shows how the actions follow each other. Depending on certain conditions the sequence of actions can vary for different executions of the activity. The example of the figure below shows an activity with three actions and two alternate routes, indicating that this activity can have two different results.



Get Book From Bookstore

An activity diagram models one activity and is made up of actions.

In principle the **activity** is the whole task that is being modeled, in the figure above being "Get Book From Bookstore". The shapes with the rounded corners are the **actions** that make up the activity. As you will see later on, an action can be detailed resulting in a new activity diagram in which the action has become the activity. In practice the term "activity" is often used to mean both, but for reasons of clarity this paper will make a distinction between the two.

DRAWING CONVENTIONS

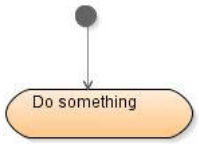
Action, Initial State, Final State

You probably are already aware that an action, or **action state** as UML officially calls it[1], is drawn as a rectangular shape with rounded corners, as in the figure on the right. Each action has a text in it, which is called the **action-expression**.

According to UML the action-expression does not need to be unique in the diagram. However, JDeveloper does not allow you creating a new action-expression with a name that already exists, but you can copy and paste an existing action. This results in a new action symbol in the diagram that has the name of the original one with "(re-use)" as a postfix, like "Do something (re-use)". In this way you can reuse the same action elsewhere in the diagram to prevent crossing transitions from cluttering the diagram.

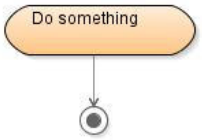
Actions, or *action states* as they officially are called, can be best named using the verb-noun construction. An activity starts with one *initial state* and ends in one or more *final states*. The states are connected with each other by *transitions*.

There is no official naming convention for action-expressions. In practice it is best to keep the name as short as possible, using the verb-noun construction, like "Request service", "Deliver order" and so on.



An activity always begins with a starting point, which UML officially calls the **initial state** and which is drawn as a solid circle. There can be only one initial state in an activity diagram, which can have only one **transition** connecting the initial state to an action as in the figure to the left.

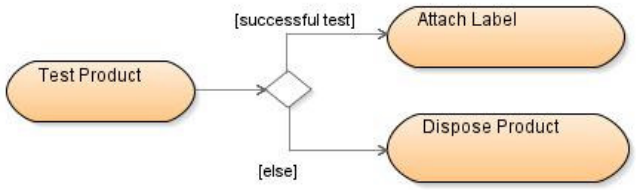
An activity has at least one ending, which is drawn by using a so-called **final state** as a solid circle with an open circle around it, like in the figure on the right. As you can see in the earlier example activity "Get Book From Bookstore", an activity can have more than one final state.



[1] Which is strange when you think of it. As the activity is the whole thing you might expect this having been called "activity state", indicating a specific state the activity is in at a certain point of time.

Decisions, Guard Conditions and Merges

In some cases the flow of control depends on a **decision** that is made, based on specific **guard conditions** or **guards** for short. A guard indicates when to follow the transition to the next action. All guard conditions of a decision should be mutually exclusive. A decision is indicated with a diamond symbol, as in the following figure.



The text of a guard is put between square brackets. In the figure a special [else] guard has been used to indicate all other, non-specified guards. The transition labeled with the [else] guard is followed when none of the other guard conditions hold.

Activities can have alternate routes. The routes are determined by *decisions* and depend on the *guards* of that decision. Alternate routes can join each other later on using a *merge*.

As you can see a decision results in alternative routes for the activity, one for each guard condition. For some activities alternative routes may join each other at some other point. For this also the diamond symbol is used, this time without guard conditions and only one transition leaving it. This is called a **merge**. Whenever alternate routes do not join, you will have more than one final state.

UML also allows another way of drawing decisions. Instead of drawing a diamond, two or more lines leave the action, each line having a guard label above it. As this way of drawing decisions is less explicit I do not recommend using it.

Synch States

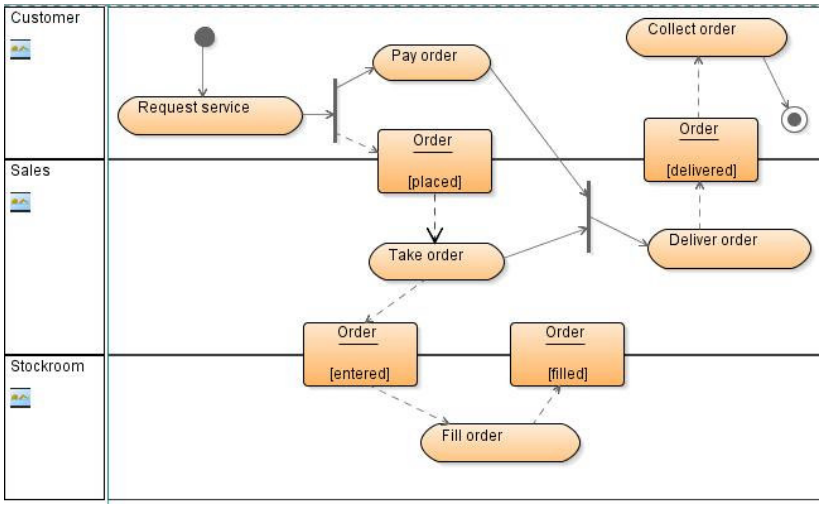
Actions can be executed asynchronously, which is modeled by using *synch states* to fork them as well as to join them later on.

For some activities there are actions that are executed asynchronously. To model this, you can use so-called **synch states** that are drawn as a thick solid line. One synch state is used to fork execution threads and another one to join them again, as in the figure below. Asynchronous threads must join at some point, but in-between a forked thread can be forked again.

Swimlanes

Actions can be organized by the entities that execute them, using *swimlanes*.

Actions can be organized by the entities that execute them (people, organizations, or other types of objects), using so-called **swimlanes**. In case of an activity diagram that details a use case, these entities typically are the actors for that use case. UML prescribes that swimlanes are drawn as vertical lanes, but JDeveloper only supports horizontal lanes, as in the following figure.



In this figure a swimlane has been created for the actors "Customer", "Sales" and "Stockroom". You are not required to use swimlanes, but in many cases using them improves the clarity of the diagram, as for each action it is clear who is responsible.

You can add swimlanes to an activity diagram by right-clicking it and then choose "Show Swimlanes" from the context-menu.

Object States and Object Flow

To model how actions manipulate an object thereby changing its state, you can add this object multiple times to the diagram, each time in a different state, as **object states**. Object states are rendered as a rectangle with the name of the object underlined and its state between brackets.

You can model changes of the state of an object by including *object states* and *object flows* in the diagram.

Transitions or **object flows** to and from object states are drawn as dashed lines, as has been done in the previous figure where the "Order" object has been included in four different states. When two actions are connected by an object state you do not also draw a transition directly between these two actions.

You can also use object states as synch states.

Subactivities

An action can be detailed by drilling it down into a lower-level activity diagram. This action is then officially called a **subactivity**.

Subactivities are indicated by the prefix "top" in the name and an icon in the bottom-right corner of the action symbol, as in the figure on the right.

An action can be drilled down in a lower-level activity diagram. The drilled down action is then called a *subactivity* .

Keep in mind that the initial state of drilled down activity diagram corresponds with the transition that enters the top action and that normally it has exactly one final state that corresponds with the transition that leaves the top action.

USING ACTIVITY DIAGRAMS

So, activity diagrams can be used in the context of the system as a whole or a subsystem to as small as an operation or a class. You can also attach activity diagrams to a use case (to model a scenario) and to collaborations (to model the dynamic aspects of a society of objects).

You can use an activity diagram to model a workflow or an operation. In both cases you can use it to model an object flow.

You typically use activity diagrams in two ways, either to model a workflow or to model an operation. In general activity diagrams that detail a use case will be of the first type. In both cases you can use the activity diagram specifically to model an object flow, showing which action(s) result in what object states. In case of a workflow the object will be a business object, whereas in case of an operation it will concern a system object.

When modeling a workflow you focus on the activities from the perspective of the actors involved. When modeling an operation you concentrate on the details of a computation or a page flow.

A short recipe for creating a workflow activity diagram (probably it's most common use), is the following:

1. Establish the focus of the activity that is going to be modeled. For non-trivial systems it will be impossible to show all interesting workflows in one diagram.
2. Select the actors that are actively involved in the activity under discussion, where "actively" means that they will perform one or more activities.
3. Create a swimlane for each of these actors.
4. Identify the preconditions of the initial state and the postconditions of the final state(s), in order to determine the boundaries of the workflow.
5. Identify the activities that make up the workflow and put them in the diagram. In case there are too many activities (being more than seven) try to collapse several actions into one (sub activity state), which is expanded in a lower-level activity diagram.
6. Draw the transitions that connect the activities, by starting with the main sequential flow (the main success scenario, in case you are detailing a use case), then add decisions and after that synch states to fork and join asynchronous actions.

7. Finally, if there are important objects involved, include them as well showing their state changes.

In case an activity diagrams is used to detail a use case, each steps of the scenario can be included in the diagram as an action. Like you can detail a step in the scenario by creating a lower-level use case for it, you can also detail the step by creating an activity diagram for its related action (subactivity). As a matter of fact, you have the option to do either one or both, whatever adds best value to making the description clear.

ORGANIZING ACTIVITY DIAGRAMS

As has been described in the white paper "Getting Started With Use Case Modeling" activity diagrams that detail a use case can be put in a sub package of the package where the use case is in, for example "mycompany.myproject.usecases.actor1.activities".

You can organize activity diagrams by putting them in a package that is a sub package of the modeling object they detail.

An activity diagram that is used to model an operation of a class can be put in a sub package of the package where the class is in, for example "mycompany.myproject.model.operationactivities". Activity diagrams for other kinds of purposes can be organized in a similar way, that is as a sub package of the modeling object that is being detailed.

The activity diagram of a subactivity can be put in a package with the name of the subactivity, for example "mycompany.myproject.usecases.actor1.activities.checkstorage".

MORE INFORMATION

The official UML site can be found at http://www.uml.org/. A short and very pragmatic reference on how to use UML modeling techniques, can be found at the site of Scott W. Ambler: http://www.agilemodeling.com/essays/umlDiagrams.htm

Other papers in the Getting Started With series are:

Getting Started With Unit-Testing
Getting Started With CVS
Getting Started With Use Case Modeling
Getting Started With Class Modeling

Getting Started With Activity Modeling
May 2007
Author: Jan Kettenis

Oracle Corporation
World Headquarters
500 Oracle Parkway
Redwood Shores, CA 94065
U.S.A.

Worldwide Inquiries:
Phone: +1.650.506.7000
Fax: +1.650.506.7200
oracle.com