

Communicating Software Design Graphically



Learning

UML

O'REILLY®

Sinan Si Albir

Learning UML

Other resources from O'Reilly

Related titles UML in a Nutshell C++ in a Nutshell
 UML Pocket Reference Java in a Nutshell

oreilly.com *oreilly.com* is more than a complete catalog of O'Reilly books. You'll also find links to news, events, articles, weblogs, sample chapters, and code examples.



oreillynet.com is the essential portal for developers interested in open and emerging technologies, including new platforms, programming languages, and operating systems.

Conferences O'Reilly & Associates brings diverse innovators together to nurture the ideas that spark revolutionary industries. We specialize in documenting the latest tools and systems, translating the innovator's knowledge into useful skills for those in the trenches. Visit *conferences.oreilly.com* for our upcoming events.



Safari Bookshelf (*safari.oreilly.com*) is the premier online reference library for programmers and IT professionals. Conduct searches across more than 1,000 books. Subscribers can zero in on answers to time-critical questions in a matter of seconds. Read the books on your Bookshelf from cover to cover or simply flip to the page you need. Try it today with a free trial.

Learning UML

Sinan Si Alhir

Table of Contents

Preface	xi
----------------------	-----------

Part I. Fundamentals

1. Introduction	3
What Is the UML?	4
The UML and Process	8
Learning the UML	13
2. Object-Oriented Modeling	15
Project Management System Requirements	15
Alphabets, Words, and Sentences	16
The Object-Oriented Paradigm	19
Paragraphs	26
Sections	39
Documents	40

Part II. Structural Modeling

3. Class and Object Diagrams	43
Classes and Objects	44
Associations and Links	55
Types, Implementation Classes, and Interfaces	69
Generalizations, Realizations, and Dependencies	72
Packages and Subsystems	79
Exercises	86

4. Use-Case Diagrams	91
Actors	92
Use Cases	94
Communicate Associations	96
Dependencies	98
Generalizations	102
Exercises	106
5. Component and Deployment Diagrams	108
Components	108
Nodes	110
Dependencies	112
Communication Associations	116
Exercises	117

Part III. Behavioral Modeling

6. Sequence and Collaboration Diagrams	121
Roles	122
Messages and Stimuli	128
Interactions and Collaborations	128
Sequence Diagrams	129
Collaboration Diagrams	138
Exercises	143
7. State Diagrams	147
States	147
Transitions	149
Advanced State Diagrams	152
Exercises	154
8. Activity Diagrams	156
Action States	156
Flow Transitions	157
Swimlanes	160
Decisions	161
Concurrency	162
Exercises	162

Part IV. Beyond the Unified Modeling Language

9. Extension Mechanisms	167
Language Architecture	167
Stereotypes	169
Properties	171
Profiles	173
Exercises	175
10. The Object Constraint Language	177
Expressions	177
Simple Constraints	180
Complex Constraints	184
Exercises	187

Part V. Appendixes

A. References	191
B. Exercise Solutions	193
Index	225

Activity Diagrams

This chapter focuses on activity diagrams, which depict the activities and responsibilities of elements that make up a system. First, I introduce activity diagrams and how they are used. Next, I discuss action states and their details. Finally, I go over flows and their details. Many details of activity diagrams that were not fleshed out in Chapter 2 are more fully elaborated here, and throughout the chapter, I include suggestions relating to activity diagrams.

Activity modeling is a specialized type of behavioral modeling concerned with modeling the activities and responsibilities of elements. You usually apply activity modeling in conjunction with sequence and collaboration modeling (Chapter 6) to explore the activities and responsibilities of interacting and collaborating elements.

Action States

As discussed in Chapter 2, as elements communicate with one another within a society of objects, each element has the *responsibility* of appropriately reacting to the communications it receives. An *action state* represents processing as an element fulfills a responsibility. There are various types of action states, including simple, initial, and final action states. The next few sections discuss these different types of action states.

Simple Action States

A *simple action state* represents processing. For example, the project management system may have the following simple action states:

Project Manager Enters Report Criteria

Indicates that the project manager enters report criteria

Project Management System Generates Report

Indicates that the project management system generates a report

Printer Prints Report

Indicates that the printer prints the report

In the UML, an action state is shown as a shape with a straight top and bottom and convex arcs on the two sides, and is labeled with the name of an operation or a description of the processing. Figure 8-1 shows the various action states associated with the project management system.

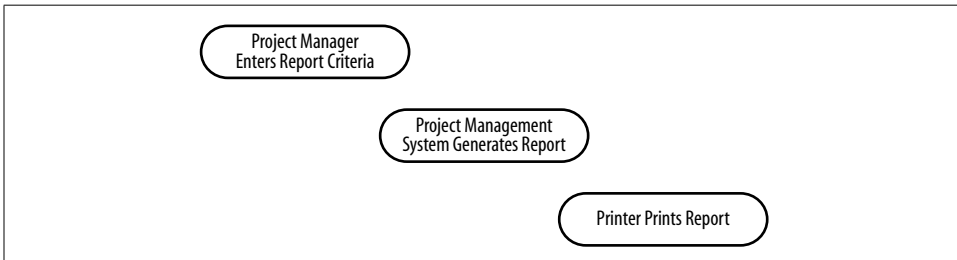


Figure 8-1. Simple action states

Initial and Final Action States

An *initial action state* indicates the first action state on an activity diagram. In the UML, an initial action state is shown using a small solid filled circle. A *final action state* indicates the last action state on an activity diagram. In the UML, a final action state is shown using a circle surrounding a small solid filled circle (a bull's eye). Figure 8-2 updates Figure 8-1 with an initial and final action state. An activity diagram may have only one initial action state, but may have any number of final action states.

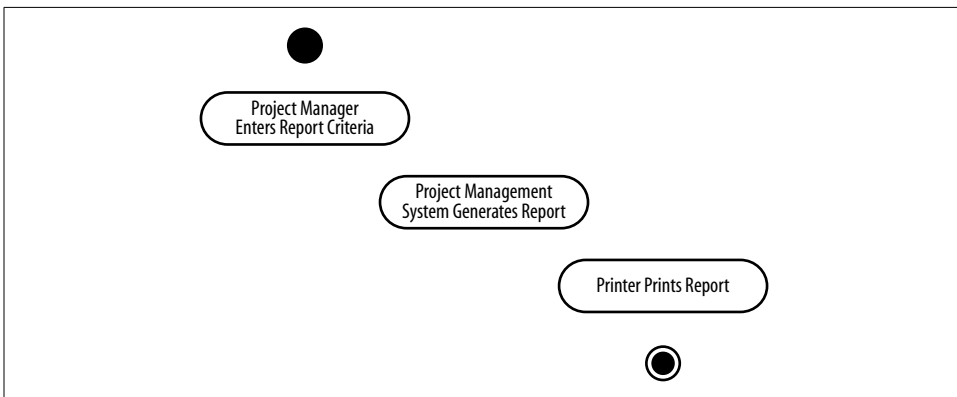


Figure 8-2. Simple, initial, and final action states

Flow Transitions

Given a collection of action states, how are those action states related to one another? Flow transitions address this issue. As discussed in Chapter 2, a *flow transition* shows how action states are ordered or sequenced. There are various types of

flow transitions, including control-flow and object-flow transitions, which are mentioned in Chapter 2 and discussed here.

Control-Flow Transitions

A *control-flow transition* indicates the order of action states. That is, once a source action state completes its processing, a target action state starts its processing. For example, the project management system may have the following order of action states for the task of generating a report:

1. The Project Manager Enters Report Criteria action state occurs first, because the project manager must enter the report criteria before the system can generate a report.
2. The Project Management System Generates Report action state occurs next, because the project management system must generate the report before the printer can print the report.
3. The Printer Prints Report action state occurs last, once the report has been generated by the project management system.

In the UML, a control-flow transition is shown as a solid line from a source action state to a target action state. Figure 8-3 shows the order of action states associated with the project management system.

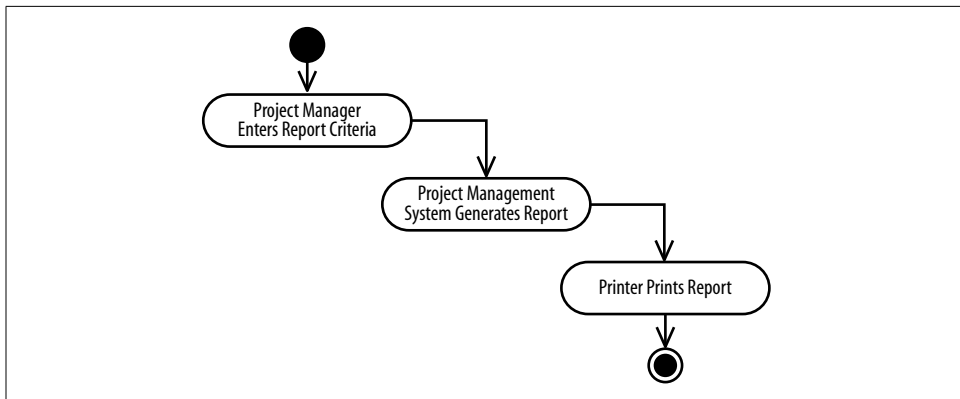


Figure 8-3. Control-flow transitions

Figure 8-3 also shows a control-flow transition originating from the initial state to indicate the first action state, and a transition to the final state to indicate the last action state.

Control-flow transitions are also known as *default transitions* or *automatic transitions*, because they are unlabeled and are immediately triggered after the source action state completes processing.

Object-Flow Transitions

An *object-flow transition* indicates that an action state inputs or outputs an object. For example, the action states shown in Figure 8-3 may have inputs and outputs as follows:

Project Manager Enters Report Criteria

Outputs a Report Criteria object and may be renamed as the Project Manager Enters Data action state

Project Management System Generates Report

Inputs the Report Criteria object and outputs a Report object and may be renamed as the Project Management System Generates Information action state

Printer Prints Report

Inputs the Report object and may be renamed as the Printer Prints Information action state

In the UML, an object-flow transition is shown as a dashed arrow between an action state and an object. An action state that uses an object as input is shown with the object-flow transition arrow pointing from the object to the action state. An action state that updates or produces an object as output is shown with the object-flow transition arrow pointing from the action state to the object. Figure 8-4 shows the objects used by the action states associated with the project management system.

Notice the new names for the action states in Figure 8-4 as compared to Figure 8-3. In Figure 8-3, the action state names conveyed some idea of the inputs and outputs from each state. Figure 8-4 however, shows these inputs and outputs explicitly, so there's no longer any need to imply them redundantly in the action state names. The advantage is the action state names can now focus purely on the actions, while the object-flow transitions indicate clearly the inputs and outputs to and from the actions.

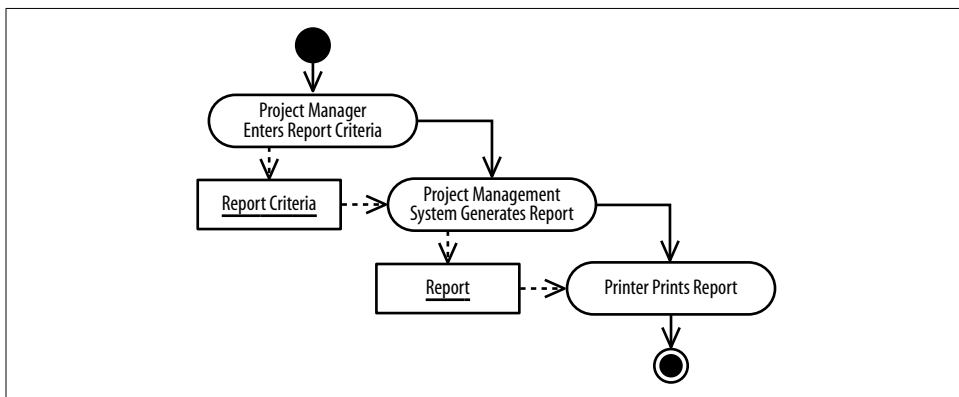


Figure 8-4. Control-flow and object-flow transitions

A control-flow transition may be omitted when an object-flow transition indicates the ordering of action states; that is, when an action state produces an output that is input to a subsequent action state, the object-flow transition implies a control-flow transition and an explicit control-flow transition is not necessary. Figure 8-5 updates Figure 8-4 by removing the unnecessary control-flow transitions.

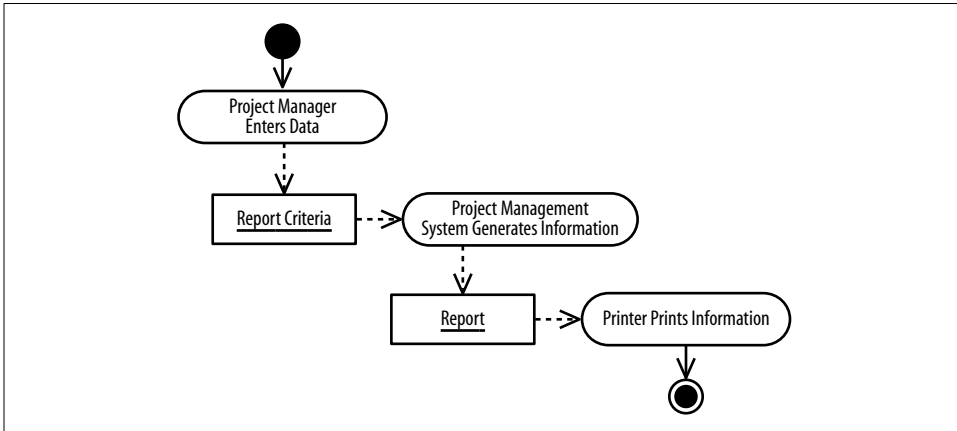


Figure 8-5. Control-flow and object-flow transitions without unnecessary control-flow transitions

Swimlanes

As discussed in Chapter 2, a *swimlane* is a visual region in an activity diagram that indicates the element that has responsibility for action states within the region. For example, the project management system may have the following swimlanes, which are illustrated in Figure 8-6:

Project Manager

Shows the action states that are the responsibility of a project manager. The swimlane makes it obvious that the project manager is responsible for entering data, thus the rather cumbersome action state name of Project Manager Enters Data may be shortened to Enter Data.

Project Management System

Shows the action states that are the responsibility of the project management system. Again, because the swimlane makes it obvious who (or what, in this case) is generating information, the rather cumbersome action state name of Project Management System Generates Information may be shortened to Generate Information.

Printer

Shows the action states that are the responsibility of a printer. Because of this swimlane, the rather cumbersome action state name of Printer Prints Information may be shortened to Print Information.

Notice how the use of swimlanes allows me to rename the action states to omit the responsible element for each action state.

In the UML, a swimlane is shown as a visual region separated from neighboring swimlanes by vertical solid lines on both sides and labeled at the top with the element responsible for action states within the swimlane. Figure 8-6 shows the swimlanes associated with the project management system.

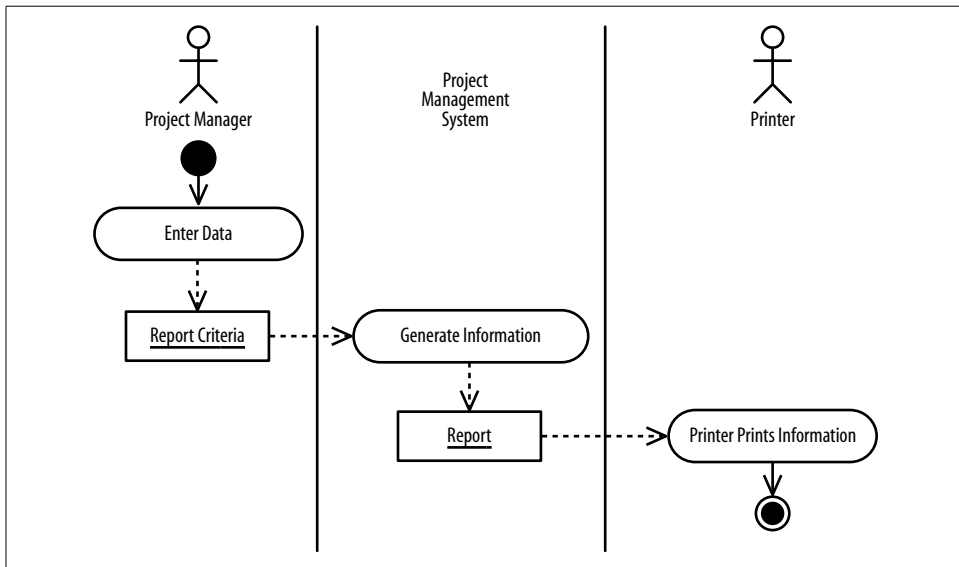


Figure 8-6. Swimlanes

Decisions

A *decision* involves selecting one control-flow transition out of many control-flow transitions based upon a condition. For example, once a report is printed by the project management system, other reports may be selected and printed if the project manager chooses to print more reports.

In the UML, a decision is shown as a diamond shape with incoming control-flow transitions and outgoing control-flow transitions where each outgoing control-flow transition is labeled with a guard condition in square brackets indicating the condition that must be satisfied for the transition to fire, or occur. Figure 8-7 shows that once a report is printed, a project manager may choose to print more reports. Notice that because the diamond shape is in the project manager's swimlane, the project manager is responsible for making the decision.

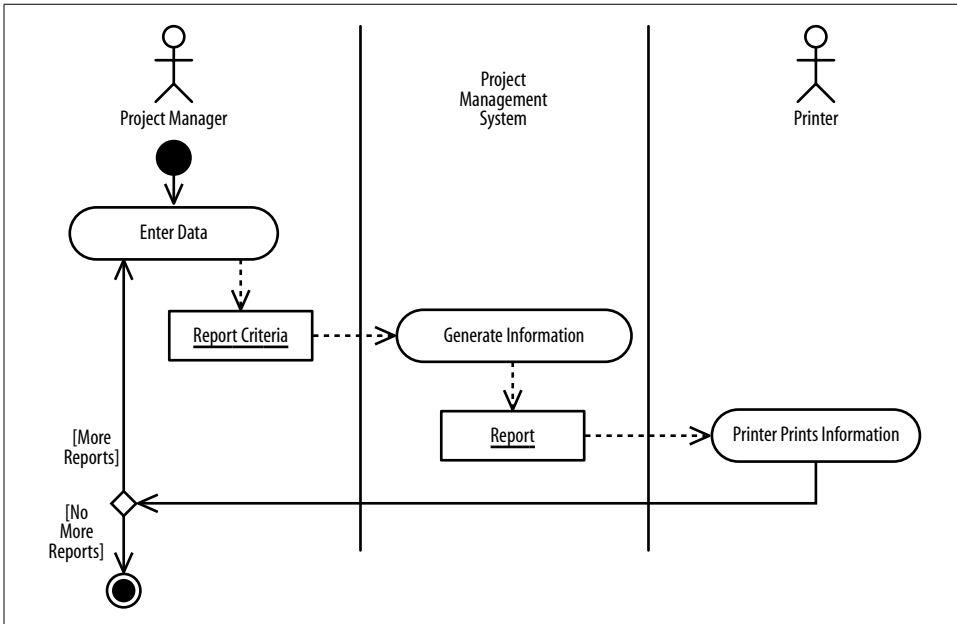


Figure 8-7. Decisions

Concurrency

Concurrency involves selecting multiple transitions simultaneously. For example, while the printer is printing a report, the printer must also monitor for other incoming print requests.

In the UML, concurrency is shown using a short heavy bar. If a bar has one incoming transition and two or more outgoing transitions, it indicates that all outgoing transitions occur once the incoming transition occurs. This is called *splitting of control*. If a bar has two or more incoming transitions and one outgoing transition, it indicates that all incoming transitions must occur before the outgoing transition occurs. This is called *synchronization of control*. Figure 8-8 shows that the printer uses concurrency to print a report using the Print Information action state, and to monitor for other incoming print requests while handling the current request using the Monitor for Print Requests action state. Once both of these action states have completed, a project manager may choose to print more reports.

Exercises

1. Describe Figure 8-9, an activity diagram that describes the action states and flow transitions between action states for a project manager printing a report using the project management system: identify action states and flow transitions.

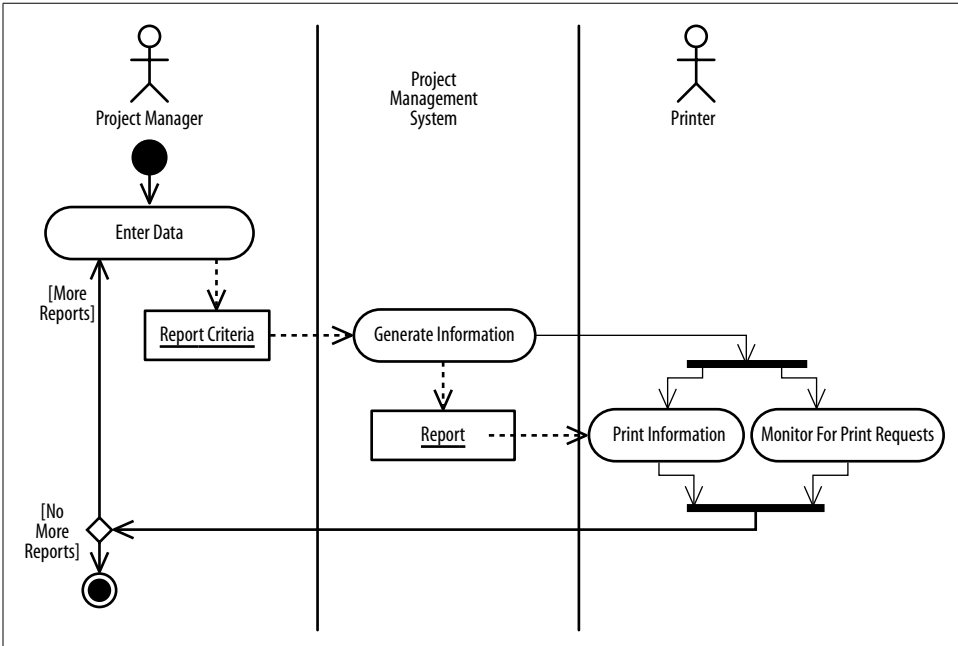


Figure 8-8. Concurrency

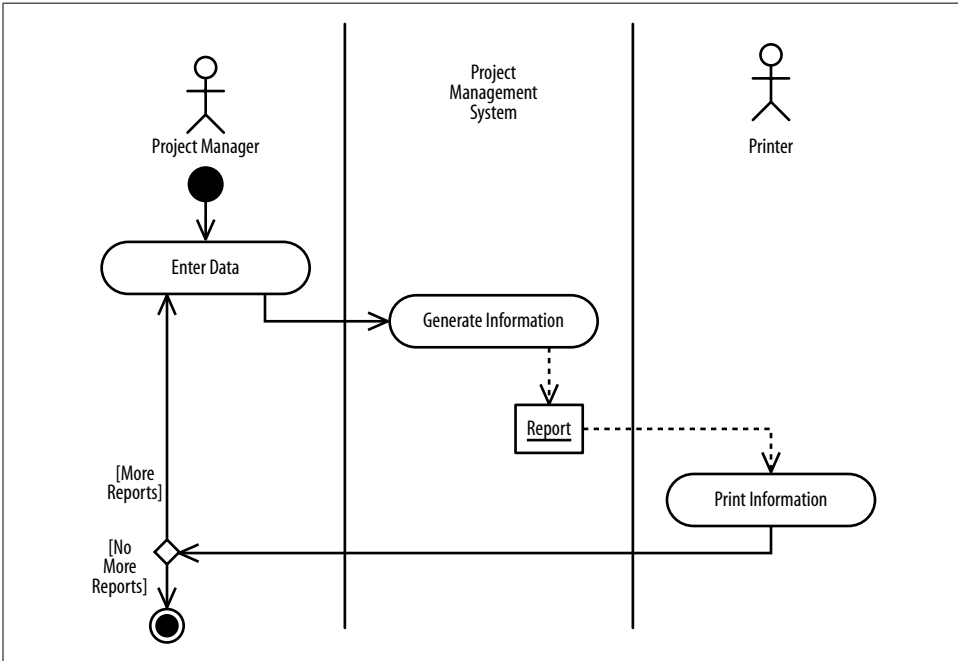


Figure 8-9. Action states and flow transitions for a project manager printing a report

2. Update Figure 8-9 stepwise to show the following details. After each step, check your answers against the solutions shown in Appendix B:
 - a. The `Enter Data` action state outputs a `Report Criteria` object that is then input by the `Generate Information` action state.
 - b. The `Generate Information` action state transitions to the `Print Information` action state only if the report is successfully generated; otherwise, the project management system generates an error using the `Generate Error` action state. In either case, the project manager may choose to print more than one report.
 - c. The project management system must simultaneously generate a report using the `Generate Information` action state (as well as handle error processing discussed in part b) and execute other processing using the `Other Processing` action state. Once the report is printed and other processing is handled, the project manager may choose to print more than one report.