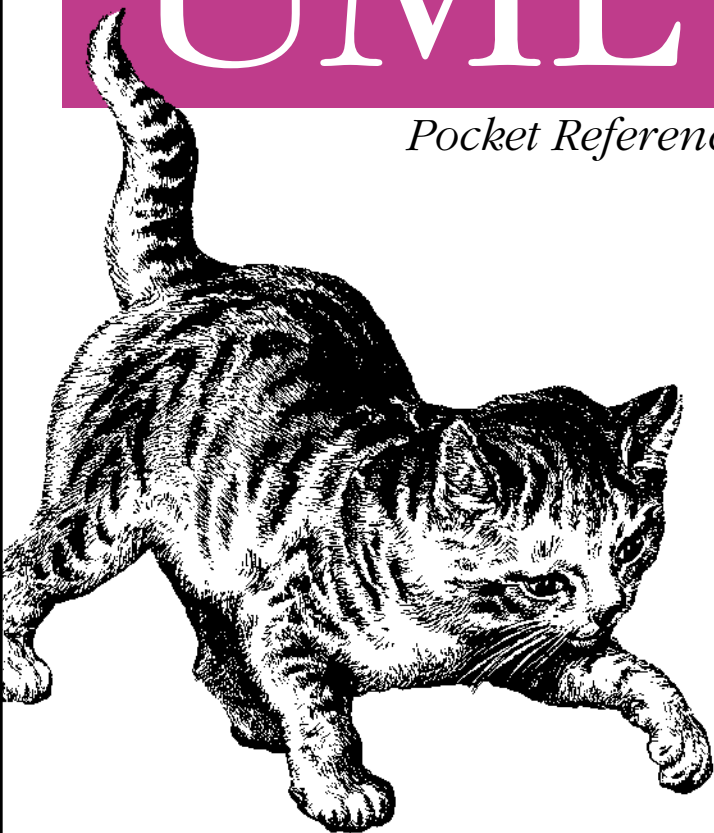


UML Syntax and Usage

UML

Pocket Reference



O'REILLY®

Dan Pilon

UML

Pocket Reference

UML

Pocket Reference

Dan Pilon

O'REILLY®

Beijing • Cambridge • Farnham • Köln • Paris • Sebastopol • Taipei • Tokyo

Use Case Diagrams

Use case diagrams are used to capture the requirements of a system. The term *use case* is often used to refer to a document that describes a particular piece of functionality a system must provide. Strictly speaking, however, a use case is a UML element, and the document describing a use case is a *use case document*. Throughout this section, the term *use case* refers to the UML notion of a use case and not to the document.

Use Cases

Typically, use cases are short phrases or sentences that sum up a distinct piece of functionality a system offers a user. Like other UML elements, use cases are often grouped into packages and can be referenced using their fully qualified name. Along with the name of a use case, there is a sequence of events that describe the behavior of the system when the use case is invoked. UML does not define a notation for recording the sequence of events for a use case, so the sequence is often described in a separate use case document, which is simply a text document created using any word-processing program. A use case is represented in a use case diagram by an ellipse, as shown in Figure 57.



Figure 57. A use case

Use cases are at a higher level of abstraction than other UML elements and describe, from the user's perspective, functionality a system must provide. Use cases do not specify how the system actually implements the functionality. Use cases are intended to communicate desired functionality from end users to project managers and actual developers.

Actors

Use cases are associated with one or more actors. An actor is a role a user takes when invoking a use case. Since a user can fulfill multiple roles, a single user can be represented by multiple actors. Likewise, a single actor can represent multiple users. An actor is represented as a stick figure with the name of the actor written underneath, as shown in Figure 58.



Figure 58. An actor

Actors do not always need to represent human users. Actors can be used to represent external systems with which a modeled system interacts. For example, you might model a robotic-tape subsystem as an actor. Actors help draw the boundary between what needs to be implemented as part of the system being modeled and what exists outside of the system.

Use cases and actors are connected using associations. When using associations on use case diagrams, the directionality of the association indicates only who initiates the interaction, not the direction of information flow. For example, Figure 59 shows both human and machine systems interacting with a use case from an ATM system. Obviously, there is interaction with the customer while the withdrawal takes place; however, the ATM never initiates a withdrawal by contacting the user.

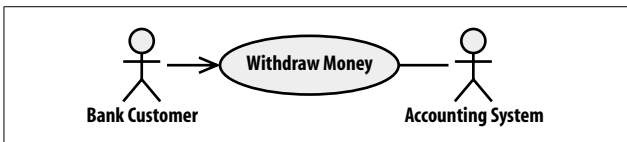


Figure 59. Human and machine actors and use cases

A bidirectional association indicates functionality that can be invoked by either the system or the actor.

Use Case Modeling

Use cases often relate to other use cases within the same system. Use cases are related using generalization, extension, or inclusion.

Use case generalization

Use case generalization behaves exactly like class generalization; a specialized use case inherits the behavior of the original use case. The specialized use case can then replace or enhance the behavior, but it adheres to the external contracts of the original use case. Generalization is modeled using the same generalization arrow used with classes, as shown in Figure 60.

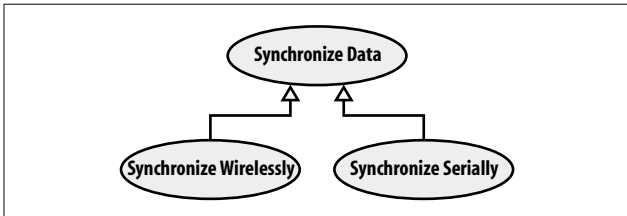


Figure 60. Use case generalization

Use case inclusion

A use case can include the behavior of another use case. The included use case is not used by itself; it can be used only in a part of a larger, separate use case. Most often, use case inclusion is used when pulling out common functionality that is shared between use cases. When including another use case, the containing use case explicitly states in its flow of events when the included use case is invoked. For example, an online purchasing system may include a use case to authenticate customers within a larger use case for purchasing an

item. In this example, customer authentication would never happen outside of the context of the larger goal.

Use case inclusion is shown using a dependency arrow that is stereotyped with `include`, as shown in Figure 61.

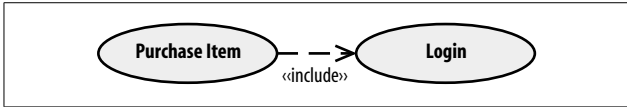


Figure 61. Use case inclusion

Use case extension

Use case extension is used to encapsulate a distinct flow of events that are not considered part of the normal or basic flow. They are not necessarily exceptional conditions, but they are sufficiently large parts of functionality that incorporating them in the base use case detracts from the focus of that use case.

When using a use case extension, the author of the base use case document explicitly states the points at which the base use case can be extended by other use cases. Unlike included use cases, extension use cases can be complete, standalone use cases that simply plug into a larger system at defined extension points in the base use case. For example, the previously mentioned online-purchasing system cannot log all communication involved in placing an order unless it is in some sort of debugging mode. The `Purchase Item` use case can be extended by a separate `Log Debugging Info` use case if the debugging criteria are met.

Use case extension is modeled using a dependency arrow stereotyped with `extend` and named with the name of the extension point, as shown in Figure 62.

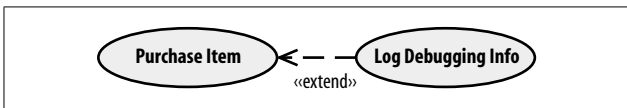


Figure 62. Use case extension

Use Case Realization

Since use cases capture requirements at a functional level, UML provides a mechanism for tracing functional requirements to their actual implementation. This mapping is called *use case realization*. Like interface realization, use case realization is shown using the realization arrow between a collaboration and a use case. A collaboration looks like a use case ellipse drawn with a dashed line, and it is typically linked to one or more UML diagrams. Figure 63 shows an example of a collaboration in a use case realization.

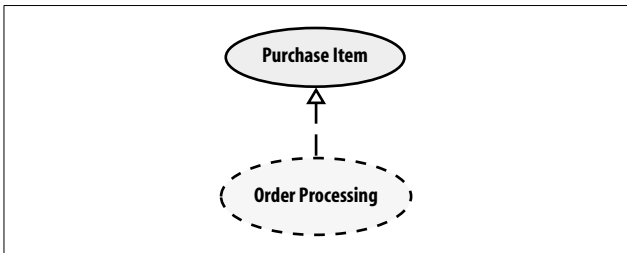


Figure 63. A use case realization

Typically, a single collaboration draws in elements from multiple packages and contains its own diagrams that show how these elements interact to provide the required functionality. Collaborations call on both static and behavioral diagrams to show how a use case is implemented. Diagrams within a collaboration often stop at subsystem or interface boundaries, in which case the details of subsystem functionality are left to subsystem modeling.

Collaborations can relate to other collaborations in that one collaboration provides more detail in a particular area than another collaboration. Thus, one collaboration may be dependent on another. To model collaboration relationships, use a dependency arrow stereotyped as *refine*. Figure 64 shows that the *Order Distribution* collaboration refines the *Order Processing* collaboration.

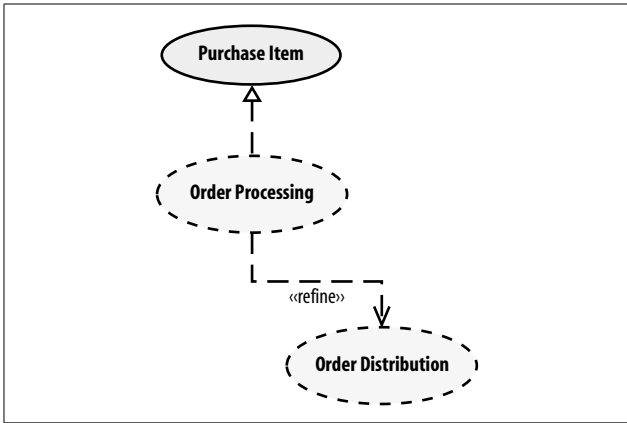


Figure 64. Collaboration refinement

Use Case Documents

While technically not part of UML, use case documents are closely related to UML use cases. A use case document is text that captures the detailed functionality of a use case. Such documents typically contain the following parts:

Brief description

Used to describe the overall intent of the use case. Typically, the brief description is only a few paragraphs, but it can be longer or shorter as needed. It describes what is considered the *happy path*—the functionality that occurs when the use case executes without errors. It can include critical variations on the happy path, if needed.

Preconditions

Conditionals that must be true before the use case can begin to execute. Note that this means the author of the use case document does not need to check these conditions during the basic flow, as they must be true for the basic flow to begin.

Basic flow

Used to capture the normal flow of execution through the use case. The basic flow is often represented as a numbered list that describes the interaction between an actor and the system. Decision points in the basic flow branch off to alternate flows. Use case extension points and inclusions are typically documented in the basic flow.

Alternate flows

Used to capture variations to the basic flows, such as user decisions or error conditions. There are typically multiple alternate flows in a single use case. Some alternate flows rejoin the basic flow at a specified point, while others terminate the use case.

Postconditions

Conditionals that must be true for the use case to complete. Postconditions are typically used by testers to verify that the realization of the use case is implemented correctly.