6 Use Case View

Overview

The use case view captures the behavior of a system, subsystem, class, or component as it appears to an outside user. It partitions the system functionality into transactions meaningful to actors—idealized users of a system. The pieces of interactive functionality are called use cases. A use case describes an interaction with actors as a sequence of messages between the system and one or more actors. The term *actor* includes humans, as well as other computer systems and processes. Figure 6-1 shows a use case diagram for a telephone catalog sales application. The model has been simplified as an example.

Actor

An actor is an idealization of a role played by an external person, process, or thing interacting with a system, subsystem, or class. An actor characterizes the interactions that a class of outside users may have with the system. At run time, one physical user may be bound to multiple actors within the system. Different users may be bound to the same actor and therefore represent multiple instances of the same actor definition. For example, one person may be a customer and a cashier of a store at different times.

Each actor participates in one or more use cases. It interacts with the use case (and therefore with the system or class that owns the use case) by exchanging messages. The internal implementation of an actor is not relevant in the use case; an actor may be characterized sufficiently by a set of attributes that define its state.

Actors may be defined in generalization hierarchies, in which an abstract actor description is shared and augmented by one or more specific actor descriptions.

An actor may be a human, a computer system, or some executable process.

An actor is drawn as a small stick person with the name below it.

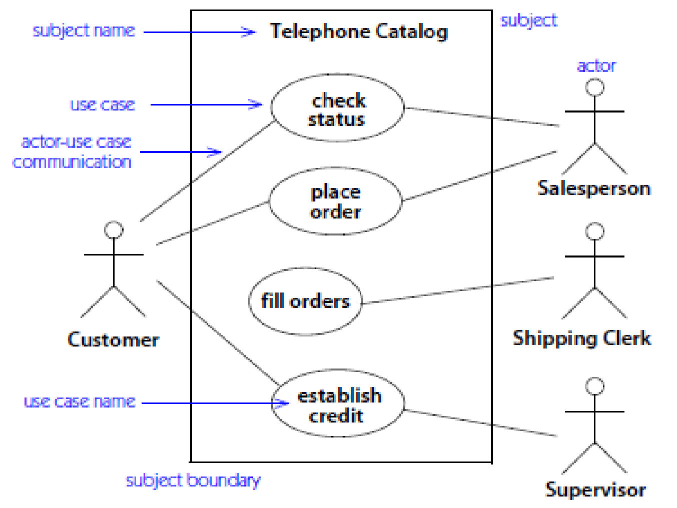

Figure 6-1. *Use case diagram*

Use Case

A use case is a coherent unit of externally visible functionality provided by a classifier (called the subject) and expressed by sequences of messages exchanged by the subject and one or more actors of the system unit. The purpose of a use case is to define a piece of coherent behavior without revealing the internal structure of the subject. The definition of a use case includes all the behavior it entails—the mainline sequences, different variations on normal behavior, and all the exceptional conditions that can occur with such behavior, together with the desired response. From the user's point of view, these may be abnormal situations. From the system's point of view, they are additional variations that must be described and handled.

In the model, the execution of each use case is independent of the others, although an implementation of the use cases may create implicit dependencies among them due to shared objects. Each use case represents an orthogonal piece of functionality whose execution can be mixed with the execution of other use cases.

The dynamics of a use case may be specified by UML interactions, shown as statechart diagrams, sequence diagrams, communication diagrams, or informal text descriptions. When use cases are implemented, they are realized by collaborations among classes in the system. One class may participate in multiple collaborations and therefore in multiple use cases.

At the system level, use cases represent external behavior of the subject as visible to outside users. A use case is somewhat like an operation, an operation invocable by an outside user. Unlike an operation, however, a use case can continue to receive input from its actors during its execution. Use cases can be applied to an entire system and can also be applied internally to smaller units of a system, such as subsystems and individual classes. An internal use case represents behavior that a subsystem presents to the rest of the system. For example, a use case for a class represents a coherent chunk of functionality that a class provides to other classes that play certain roles within the system. A class can have more than one use case.

A use case is a logical description of a slice of functionality. It is not a manifest construct in the implementation of a system. Instead, each use case must be mapped onto the classes that implement a system. The behavior of the use case is mapped onto the transitions and operations of the classes. Inasmuch as a class can play multiple roles in the implementation of a system, it may therefore realize portions of multiple use cases. Part of the design task is to find implementation classes that cleanly combine the proper roles to implement all the use cases, without introducing unnecessary complications. The implementation of a use case can be modeled as a set of one or more collaborations. A collaboration is a realization of a use case.

A use case can participate in several relationships, in addition to association with actors (Table 6-1).

Table 6-1: *Kinds of Use Case Relationships*

Relationship	*Function*	*Notation*
association	The communication path between an actor and a use case that it participates in	———
extend	The insertion of additional behavior into a base use case that does not know about it	«extend» - - - ->
include	The insertion of additional behavior into a base use case that explicitly describes the insertion	«include» - - - ->
use case generalization	A relationship between a general use case and a more specific use case that inherits and adds features to it	———▷

A use case is drawn as an ellipse with its name inside or below it. It is connected by solid lines to actors that communicate with it.

The description of a large use case can be factored into other, simpler use cases. This is similar to the way the description of a class can be defined incrementally from the description of a superclass. A use case can incorporate the behavior of other use cases as fragments of its own behavior. This is called an include relationship. The included use case is not a specialization of the original use case and cannot be substituted for it.

A use case can also be defined as an incremental extension to a base use case. This is called an extend relationship. There may be several extensions of the same base use case that may all be applied together. The extensions to a base use case add to its semantics; it is the base use case that is instantiated, not the extension use cases.

The include and extend relationships are drawn as dashed arrows with the keyword «include» or «extend». The include relationship points at the use case to be included; the extend relationship points at the use case to be extended.

A use case can also be specialized into one or more child use cases. This is use case generalization. Any child use case may be used in a situation in which the parent use case is expected.

Use case generalization is drawn the same as any generalization, as a line from the child use case to the parent use case with a large triangular arrowhead on the parent end. Figure 6-2 shows use case relationships in the catalog sales application.

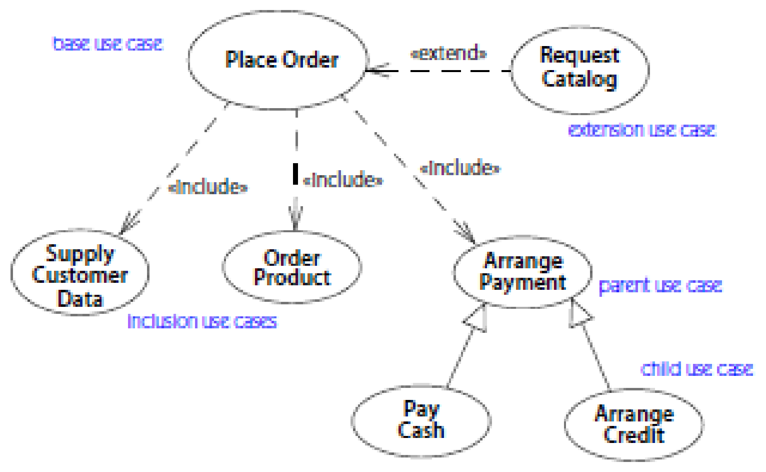

Figure 6-2. *Use case relationships*