

# Creating Use Case Diagrams

---

## Objectives

Upon completion of this module, you should be able to:

- Justify the need for a Use Case diagram
- Identify and describe the essential elements in a UML Use Case diagram
- Develop a Use Case diagram for a software system based on the goals of the business owner
- Develop elaborated Use Case diagrams based on the goals of all the stakeholders
- Recognize and document use case dependencies using UML notation for extends, includes, and generalization
- Describe how to manage the complexity of Use Case diagrams by creating UML packaged views

## Additional Resources

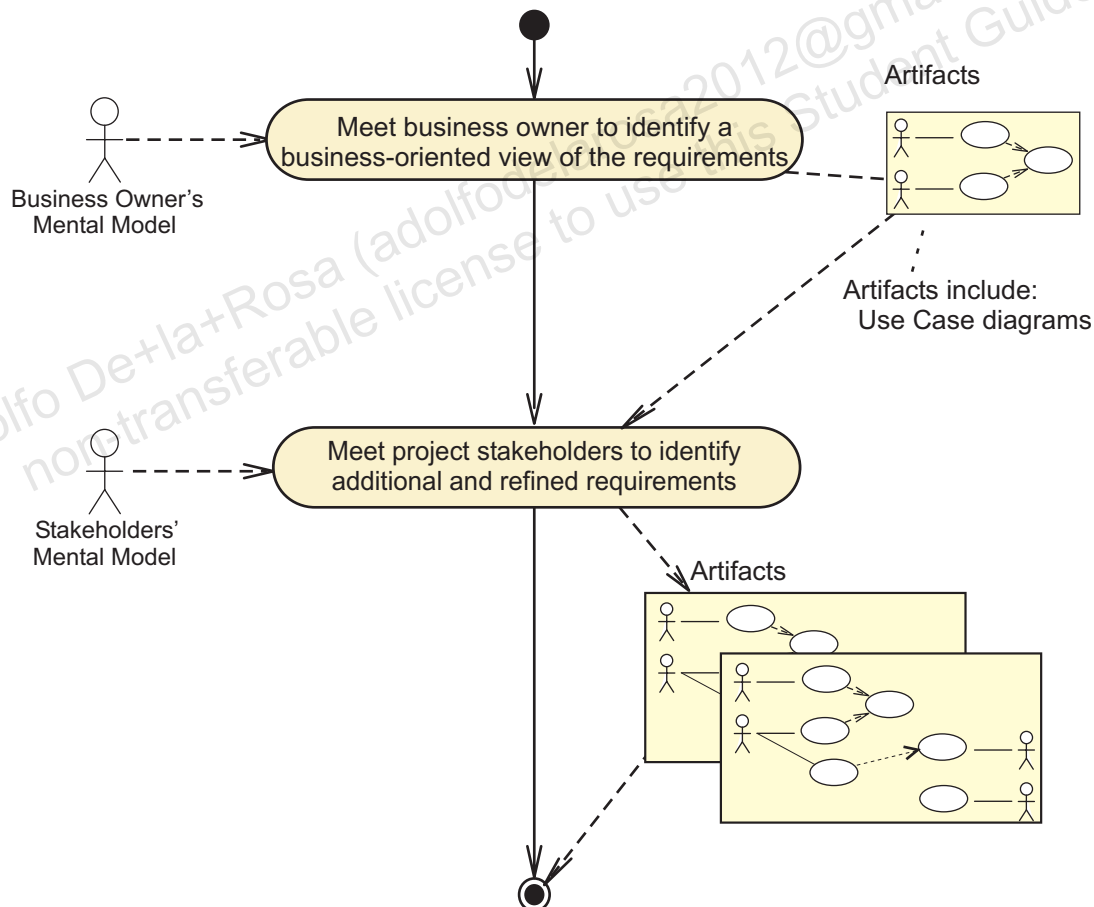
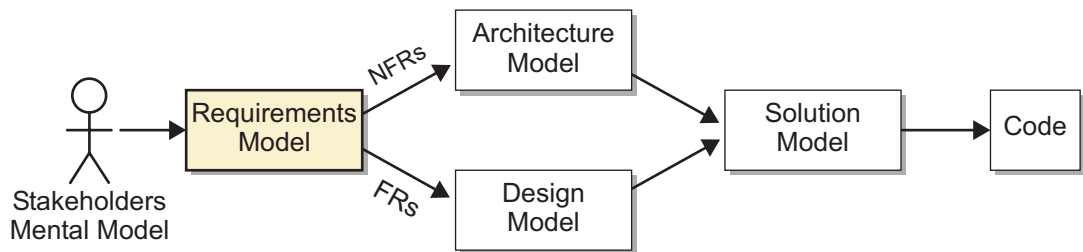


**Additional resources** – The following references provide additional information on the topics described in this module:

- Booch, Grady, James Rumbaugh, Ivar Jacobson. *The Unified Modeling Language User Guide*. Reading: Addison Wesley Longman, Inc., 1999.
- Folwer, Martin, Kendall Scott. *UML Distilled (2nd ed)*. Reading: Addison Wesley Longman, Inc., 2000.
- The Object Management Group. “Unified Modeling Language (UML), Version 2.2”  
[<http://www.omg.org/technology/documents/formal/uml.html>].
- Rosenberg, Doug, Kendall Scott. *Use Case Driven Object Modeling with UML (A Practical Approach)*. Reading: Addison Wesley Longman, Inc., 1999.
- Larman, Craig. *Applying UML and Patterns (3rd ed)*. Upper Saddle River: Prentice Hall, 2005.
- Jacobson, Ivar, Grady Booch, James Rumbaugh. *The Unified Modeling Language Reference Manual (2nd ed)*. Addison-Wesley, 2004.

# Process Map

This module covers the next step in the Requirements Gathering workflow: Creating the initial Use Case diagram. Figure 3-1 shows the activity and artifact covered in this module.



**Figure 3-1** Requirements Gathering Process Map

## Justifying the Need for a Use Case Diagram

This module describes how to use a Use Case diagram to model and document the goals (use cases) of its users. The following points explain the purpose of a Use Case diagram:

- A Use Case diagram enables you to identify—by modeling—the high-level functional requirements (FRs) that are required to satisfy each user's goals.

A Use Case diagram provides a visual representation of the high-level FRs. The Use Case diagram is often easier to model with the client-side stakeholders than the alternative textual representation.

- The client-side stakeholders need a big-picture view of the system.

A Use Case diagram provides a high-level view of the entire system. All non-trivial systems will have too many use cases to view at the same time. The use of UML packages allows you to see a high-level view of the packages, each containing either use cases or subpackages. The packages are simply views of related use cases that can be categorized in different ways—for example, Sales, Marketing, and Shipping. This view can be the basis of a common language between the client-side stakeholders and the development team.

- The use cases form the basis from which the detailed FRs are developed.

Use cases are the central focus of system development. A Use Case diagram is the high-level guide for the development team. The lower-level functionality of the system can be identified by exploring the internal behavior of each use case, which will be covered in the following module.

- Use cases can be prioritized and developed in order of priority.

Use cases can be assigned priorities based on business need, complexity, and dependency on other use cases. In an incremental development process, the priority affects the iteration in which the use case will be developed.

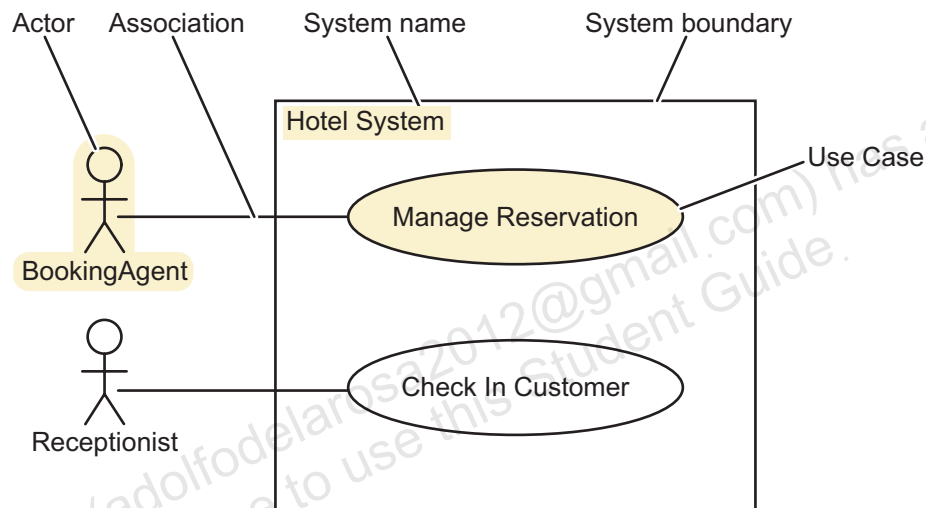
- Use cases often have minimal dependencies, which enables a degree of independent development.

Because use cases often have minimal dependencies, they can be developed in parallel or by more specialist teams.

## Identifying the Elements of a Use Case Diagram

A Use Case diagram shows the relationships between actors and the goals they wish to achieve.

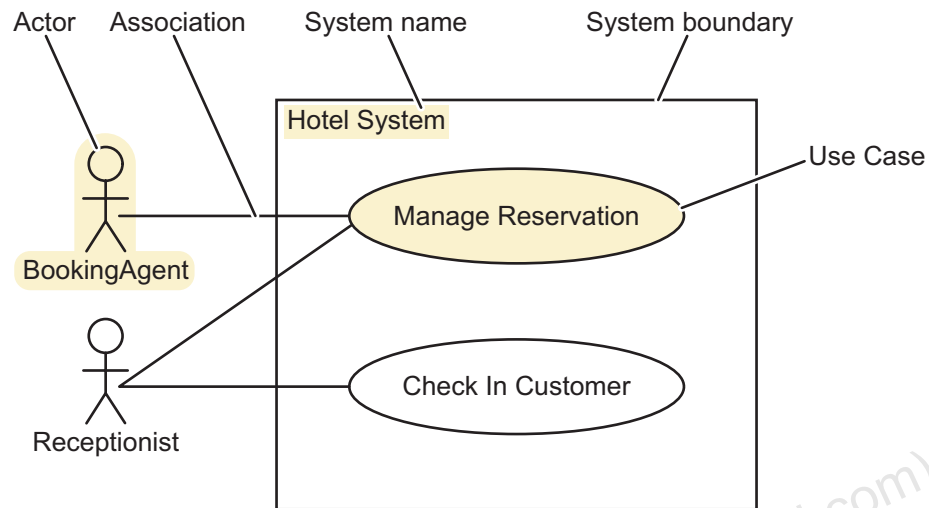
A *Use Case diagram* provides a visual representation of the system, the use cases that the system provides, and the actors (job roles) that use the system to perform specific functions. Figure 3-2 shows an example Use Case diagram.



**Figure 3-2** An Example Use Case Diagram

An alternative style is to draw an association line between the Receptionist actor and the Manage Reservation use case as the Receptionist can manage reservations as well.

Actors are simply roles. Any physical person, system, or device can assume multiple roles. Therefore, the job title Receptionist can assume the Receptionist role and also the Booking Agent role. If this was not the case, then the Duty Manager actor would need associations to most of the use cases in the hotel, as would the Hotel Manager and other actors.



**Figure 3-3** An Example Alternate Style Use Case Diagram

## Actors


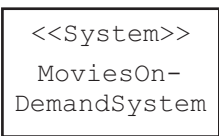

“An actor is a role that a user plays with respect to the system.”  
(Fowler UML Distilled page 42)

“An Actor models a type of role played by an entity that interacts with the subject (e.g., by exchanging signals and data), but which is external to the subject (i.e., in the sense that an instance of an actor is not a part of the instance of its corresponding subject). Actors may represent roles played by human users, external hardware, or other subjects. Note that an actor does not necessarily represent a specific physical entity but merely a particular facet (i.e., “role”) of some entity that is relevant to the specification of its associated use cases. Thus, a single physical instance may play the role of several different actors and, conversely, a given actor may be played by multiple different instances. (UML Superstructure Specification, v2.2)

**Note** – The subject is the system under consideration to which the use cases apply.

Anyone or anything that is external to the system and interacts with the system is an *actor*. There are fundamentally three classes of actors: people, external systems or devices, and time. Figure 3-4 illustrates these three actor types.

**Table 3-1**

 BookingAgent	 <<System>> MoviesOn- DemandSystem	 Time
<p>This icon represents a human actor (user) of the system.</p>	<p>This icon can represent any actor, but is usually used to represent external systems, devices, or time.</p>	<p>This icon represents a time-trigger mechanism that activates a use case.</p>

**Figure 3-4** Actor Types

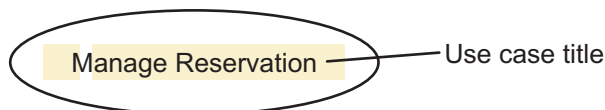
- Actors that initiate and control the use case are subcategorized as primary actors.  
These actors will participate for the entire duration of the use case.
- Actors that are used by a use case are subcategorized as secondary actors.  
These actors will generally participate for only part of the duration of the use case.



## Use Cases

A use case is “The specification of a sequence of actions, including variants, that a system can perform.”

A *use case* describes an interaction between an actor and the system to achieve a goal. Figure 3-5 shows an example use case.



**Figure 3-5** An Example Use Case

- A use case encapsulates a major piece of system behavior with a definable outcome.

A use case provides a visual encapsulation of all of the detailed actions involved in a major system behavior. The use case title provides a communication tool for stakeholders to discuss the system at a high-level.

- A use case is represented as an oval with the use case title in the center.

Alternatively, the title can be placed under the oval.




---

**Note** – Use case title and use case name are synonymous.

---

- A good use case title should consist of a brief but unambiguous verb-noun pair.
  - “Check In” has a verb, but no noun.
  - “Check In Customer when they arrive at the hotel” is too detailed.
  - In tennis, “Enter Score” is a verb-noun pair. However, “Enter Final Match Score” or “Enter Current Set Score” would reduce the ambiguity.

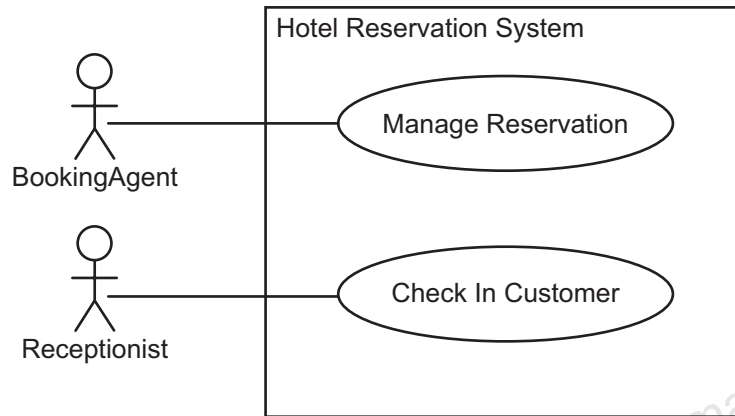
If a verb-noun pair is ambiguous, you will need to use a verb-noun pair phrase to reduce the ambiguity. However, it should be as brief as possible. Also, the terms used should be included in the glossary of terms.

- Use case internal descriptions can be written in a User Interface (UI) independent form.

This allows use cases to be more re-usable and accommodate new UIs. It also allows you to focus on the use case's business process instead of how the UI works. However, this approach might not always be suitable.

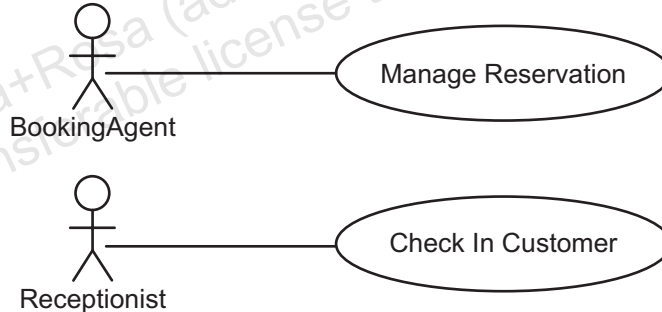
## System Boundary

A Use Case diagram is usually shown with a system boundary. Figure 3-6 illustrates the system boundary.



**Figure 3-6** A Use Case Diagram With a System Boundary

A Use Case diagram may also be drawn without a system boundary. Figure 3-7 illustrates an example of a Use Case diagram without a boundary.



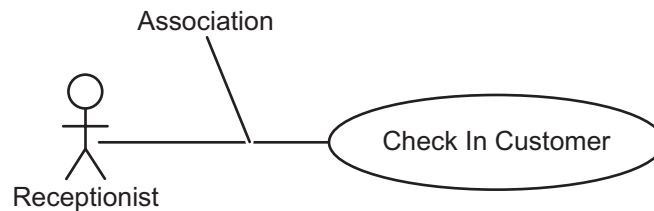
**Figure 3-7** A Use Case Diagram Without a System Boundary

For clarity, you should use a system boundary when drawing a Use Case diagram.

## Use Case Associations

A use case association represents “the participation of an actor in a use case.” (UML v1.4 spec. page 357)

In the Hotel Reservation System, a receptionist role uses the System to perform the Check In Customer use case. Figure 3-8 shows this association.



**Figure 3-8** An Example Use Case Association

- An actor must be associated with one or more use cases.  
An actor with no use case associations does not interact with the system, so there is no reason to represent the actor in the diagram.
- A use case must be associated with one or more actors.  
However, later sections in this module describe that other use cases might be identified that are included in the activity of an essential use case; these sub-use cases might not be directly associated with an actor because they implicitly interact with the included (sub) use case.
- An association is represented by a solid line usually with no arrowheads.  
Some UML tools put an arrow on one end of the association line by default. Some people use this arrow notation to distinguish between the interactions initiated by a primary actor and those initiated by a use case on a secondary actor.

**Note** – Models evolve. It is likely that your first attempt in creating a Use Case diagram of a system will not be final. As the system evolves (through iterative development), you will need to update and refine the Use Case diagram.



## Creating the Initial Use Case Diagram

One of the primary aims of the initial meeting with the project's business owner is to identify the business-significant use cases.

- A use case diagram may be created during the meeting  
This may achieve faster and more accurate results by using the benefits of UML modeling to assist in identifying the major business-oriented goals that the system will achieve.
- Alternatively, the diagrams can be created after the meeting from textual notes.  
The goals can be identified and listed, usually in textual form, during the meeting.

The following text provides an abstract from the meeting with the business owner:

The booking agent (internal staff) must be able to manage reservations on behalf of customers who telephone or e-mail with reservation requests. The majority of these requests will make a new reservation, but occasionally they will need to amend or cancel a reservation. A reservation holds one or more rooms of a room type for a single time period, and must be guaranteed by either an electronic card payment or the receipt of a purchase order for corporate customers and travel agents. These payment guarantees must be saved for future reference.

A reservation can also be made electronically from the Travel Agent system and also by customers directly via the internet.

The receptionist must be able to check in customers arriving at the hotel. This action will allocate one or more rooms of the requested type. In most cases, a further electronic card payment guarantee is required.

Most receptionists will be trained to perform the booking agent tasks for customers who arrive without a booking or need to change a booking.

The marketing staff will need to manage promotions (special offers) based on a review of past and future reservation statistics. The marketing staff will elaborate on the detailed requirements in a subsequent meeting.

The management need a daily status report, which needs to be produced when the hotel is quiet. This activity is usually done at 3 a.m.

Figure 3-9 shows an initial Use Case diagram for these high-level requirements.

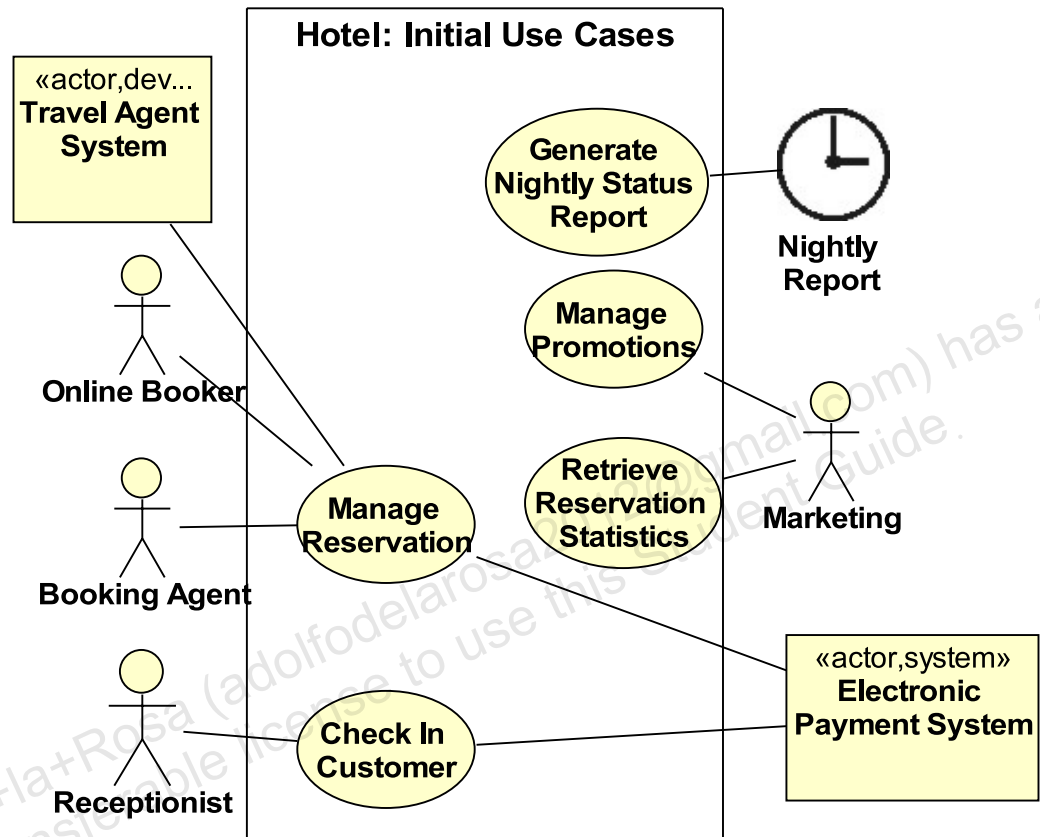


Figure 3-9 Partial Initial Use Case Diagram for a Hotel

## Identifying Additional Use Cases

During the meeting with the business owner, you will typically discover 10 to 20 percent of the use case titles needed for the system.

During the meeting with the other stakeholders, you will discover many more use case titles that you can add to the diagram. For example:

- Maintain Rooms
  - Create, Update, and Delete
- Maintain RoomTypes
  - Create, Update, and Delete

The time of discovery of use cases depends upon the development process.

- In a non-iterative process:
  - You ideally need to discover all of the remaining use case titles, bringing the total to 100 percent.
  - However, this is a resource-intensive task and is rarely completely accurate.
- In an iterative/incremental development process, an option is to:
  - Discover a total of 80 percent of the use case titles in the next few iterations for 20 percent of the effort. This is just one of the many uses of the 80/20 rule.
  - Discover the remaining 20 percent of use case titles in the later iterations for minimal effort.

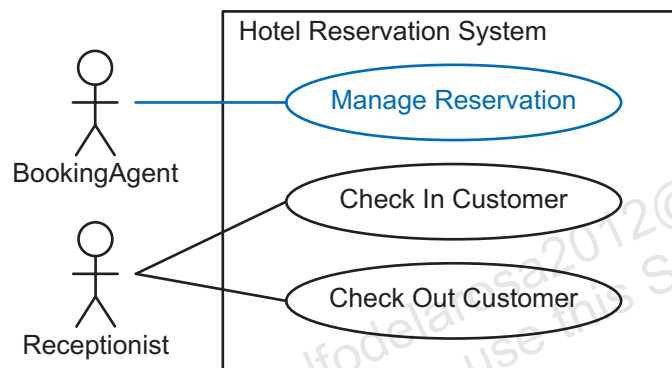
This process works well with software that is built to accommodate change.

The 80/20 rule originates from Pareto Principle, where Vilfredo Pareto created a formula describing the unequal distribution of wealth where 20 percent of the population owned 80 percent of the wealth. This rule applies to many other areas. For example, you can discover 80 percent of the requirement for 20 percent of the effort.

## Use Case Elaboration

During the meeting with the other stakeholders, you will discover many more use cases that you can add to the diagram. You might also find that some use cases are too high-level. That is, a use case might describe a business function that includes several related workflows.

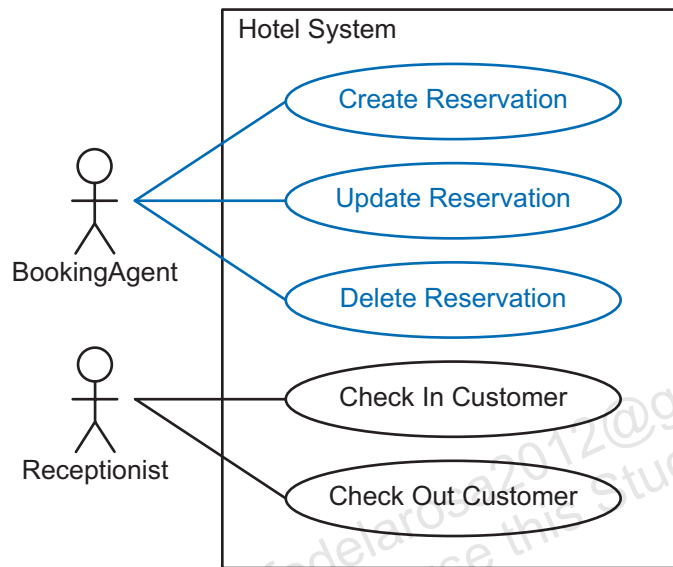
For example, the Manage Reservation use case is too high-level because it represents several different workflows that all deal with hotel reservations. Figure 3-10 shows this.



**Figure 3-10** An Example High-Level Use Case



In these situations, it is useful to introduce new use cases that separate the workflows. For example, the Manage Reservation use case can be separated into three workflows: one for creating a new reservation, one for updating an existing reservation, and one for deleting an existing reservation. Figure 3-11 shows this.



**Figure 3-11** The High-Level Use Case Separated Into Individual Workflows

Typically, *managing an entity* implies being able to Create, (Retrieve), Update, and Delete an entity (so called, CRUD operations). Other keywords include:

- Maintain
- Process

Other high-level use cases can occur. Identify these by analyzing the use case scenarios and look for significantly divergent flows. Also, if several scenarios have a different starting point, these scenarios might represent different use cases.

Be careful not to refine the use cases too much because use case refinement can become an exercise in functional decomposition and can result in *analysis paralysis*.

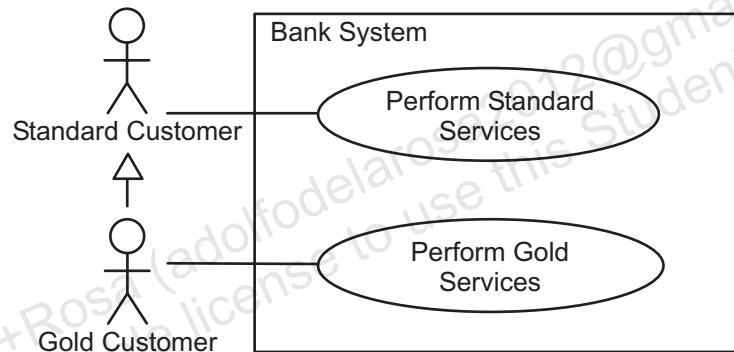
## Analyzing Inheritance Patterns

Inheritance can occur in Use Case diagrams for both actors and use cases:

- An actor can inherit all of the use case associations from the parent actor.
- A use case can be *subclassed* into multiple, specialized use cases.

### Actor Inheritance

An actor can inherit all of the use case associations from the parent actor. For example, the Gold Customer *is a kind of* Standard Customer but can access additional use cases. Therefore, you can use actor inheritance to represent this relationship. Figure 3-12 illustrates this.

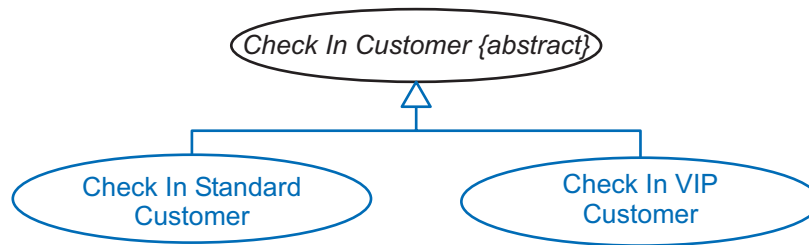


**Figure 3-12** Example Actor Inheritance

This relationship is often used when the phrase “*is a kind of*” does not accurately describe the relationship between the actors. This can often present problems later on, just as it would with class inheritance. For example, if you later (in a later iteration or later version of the software) discover a use case where the Standard Customer can perform a use case that the Gold Customer cannot, you will have to go back and remove this inheritance. This action might require you to make major changes to the software system.

## Use Case Specialization

A use case can be *subclassed* into multiple, specialized use cases. For example, the use case of Retrieve a Reservation can be performed in two distinct ways: search by ID or search by customer. Figure 3-13 illustrates this.



**Figure 3-13** Example Use Case Specialization

Use case specializations are *usually* identified by significant variations in the use case scenarios. These variations are often due to different functional strategies for accomplishing the use case.

The base use case may be marked as abstract, in which case you cannot instantiate the base use case. Therefore in the example in Figure 3-13, you can only instantiate Check In Standard Customer or Check In VIP Customer. If Check In Customer was not abstract, you could instantiate that use case as well.

Using this notation can make the writing of Use Case forms more complicated, therefore is often avoided.

## Analyzing Use Case Dependencies

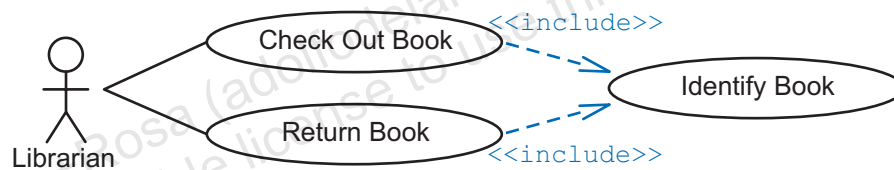
Use cases can depend on other use cases in two ways:

- One use case (a) *includes* another use case (i).  
This means that the one use case (a) requires the behavior of the other use case (i) and always performs the included use case.
- One use case (e) can *extend* another use case (b).  
This means that the one use case (e) can (optionally) extend the behavior of the other use case (b).

### The «include» Dependency

The include dependency enables you to identify behaviors of the system that are common to multiple use cases.

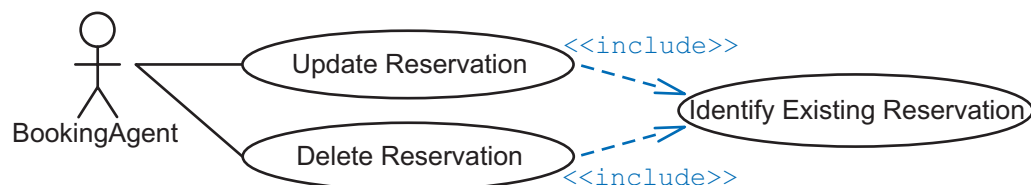
This dependency is drawn with a dependency arrow, and includes the «include» stereotype label. This is illustrated in Figure 3-14.



**Figure 3-14** Example «include» Dependency

To identify common behavior, review the scenarios of multiple use cases for common behaviors. Give this behavior a name and place it in the Use Case diagram with an «include» dependency.

For example, both the Update Reservation and Delete Reservation use cases require the ability to identify the reservation that is being updated or deleted. This would be done if there is significant actor interaction. This behavior can be given a name “Identify Existing Reservation” and placed in the diagram with the appropriate «include» dependencies. Figure 3-15 shows this.

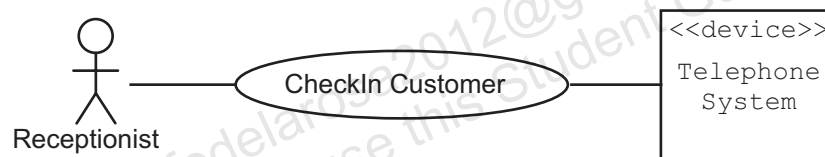


**Figure 3-15** An «include» Dependency in the Hotel Reservation System

Another situation for refining the Use Case model involves interactions with external systems. It is often useful to explicitly represent these interactions with separate use cases.

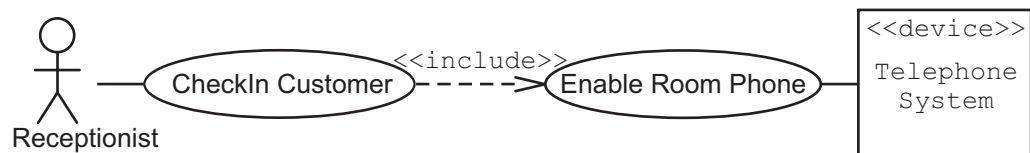
To identify behavior associated with an external actor (system or device), review the scenarios for sequences of behavior that involve an external actor (system or device). Give this behavior a name, and place it in the Use Case diagram with an «include» dependency. Look for other use cases that might include these use cases and record additional dependencies.

For example, the Hotel Reservation System will be connected to an external Hotel Phone System. This system is used when a customer checks in to enable the phone in the room that the customer will occupy. Figure 3-16 shows the Use Case diagram without this «include» dependency. As a result, the reason for the participation of the Telephone System is less obvious.



**Figure 3-16** Example Where the Secondary Actor's Role is not Clear

Figure 3-17 shows the secondary actor's reason for participation in this use case more clearly.

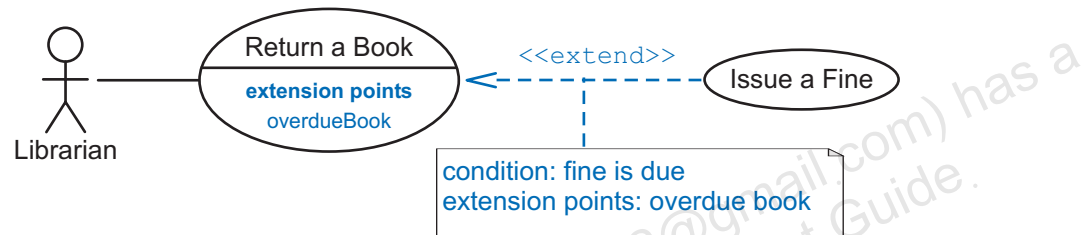


**Figure 3-17** Example Using «include» to Highlight the Secondary Actor's Participation

## The «extend» Dependency

The extend dependency enables you to identify behaviors of the system that are not part of the primary flow, but exist in alternate scenarios.

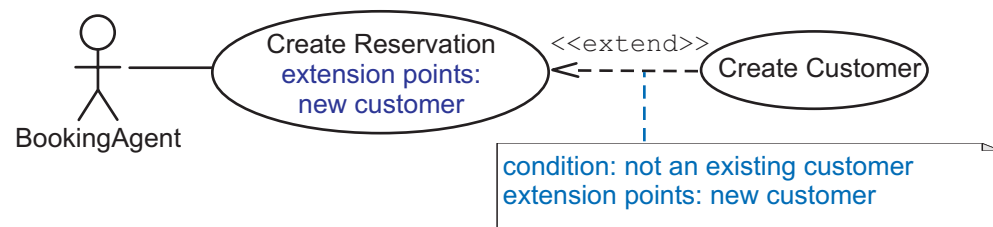
This dependency is drawn with a dependency arrow, an «extend» stereotype label, and an additional label that identifies the “extension point.” The extension point defines the condition within the main use case under which the extension use case is required. Furthermore, the extension points can be listed in the use case node itself. Figure 3-18 illustrates this.



**Figure 3-18** Example Extends Dependency

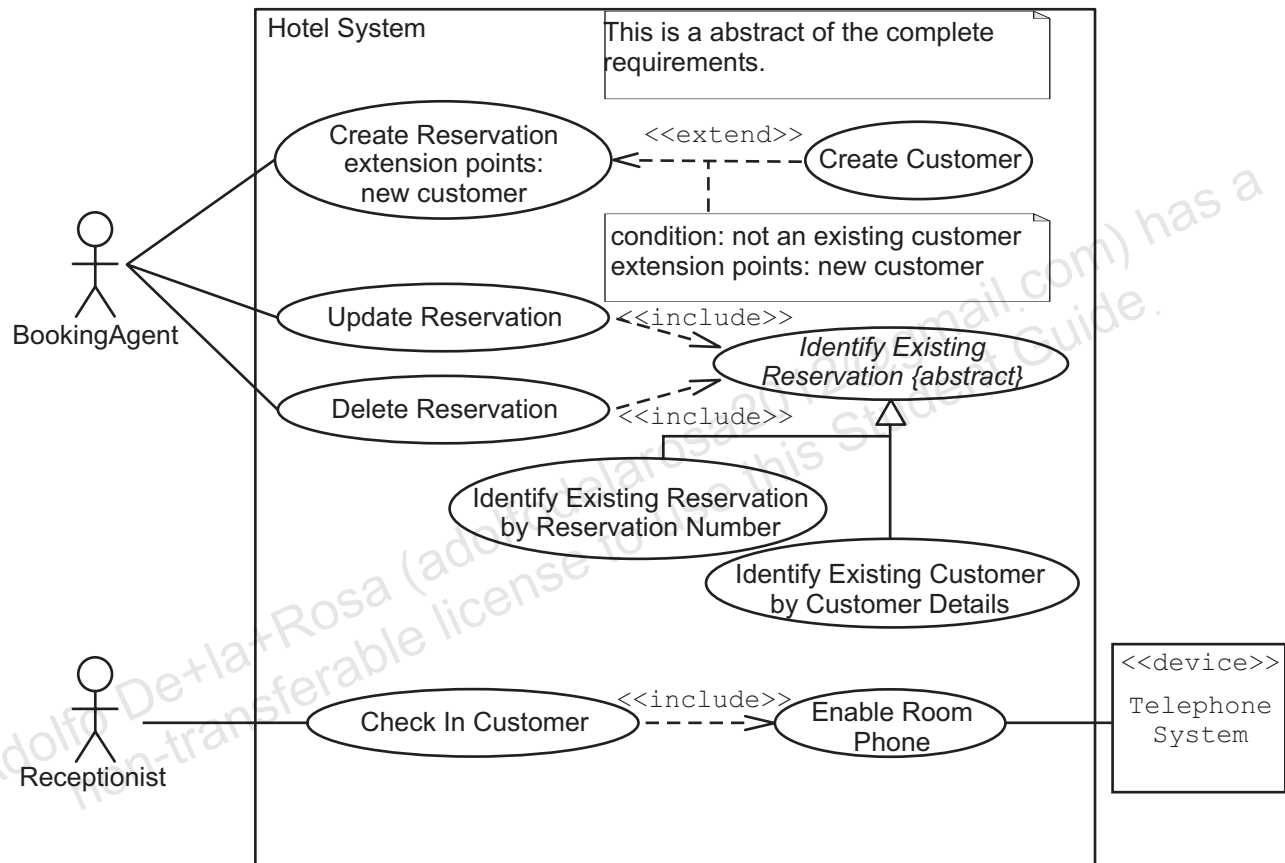
To identify behaviors associated with an alternate flow of a use case, review the scenarios for sequences of behavior that are optional or could be added in a later increment of the system. Give this behavior a name and place it in the Use Case diagram with an «extend» dependency. Look for other use cases that might be extended by these use cases.

For example, when a reservation is created, the reservation must be associated with a customer. This customer can be an existing customer, which is simply a step in the main or alternate flow. However, if the customer is not an existing customer, you need to perform additional behavior to add the customer. Figure 3-19 shows this.



**Figure 3-19** An «extend» Dependency in the Hotel Reservation System

Figure 3-20 illustrates an example showing a possible refinement of the Use Case diagram for the Hotel Reservation System. In this example, the Identify Existing Reservation by Reservation Number and Identify Existing Customer by Customer Details use cases will have different interactions with the actor. However, if these differences are minor, there might be no need to provide these two specialization use cases.

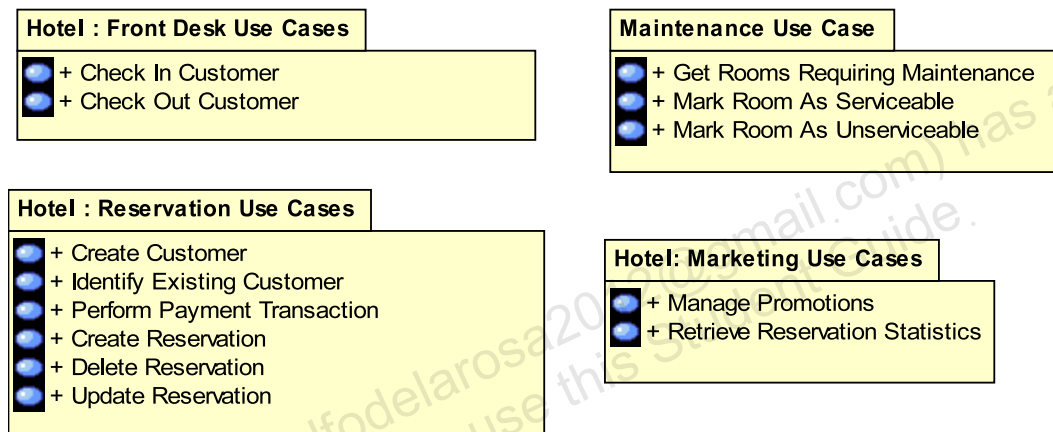


**Figure 3-20** A Combined Example From the Hotel Reservation System

## Packaging the Use Case Views

It should be apparent that any non-trivial software development would need more use cases than could be viewed at one time. Therefore, you need to be able to manage this complexity.

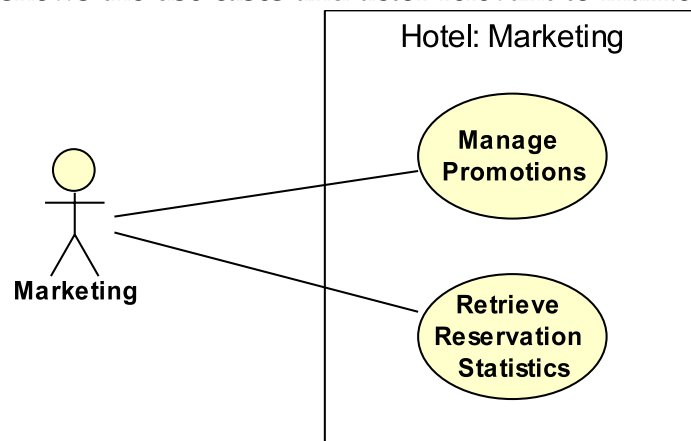
One way of managing this complexity is to break down the use cases into package views. Figure 3-21 shows some of the packages you could create in order to view subsets of the use cases.



**Figure 3-21** Example Showing Packages of Use Case

You can look inside each package to reveal the detailed content. Also, a use case element may exist in multiple packages, where it participates in multiple views.

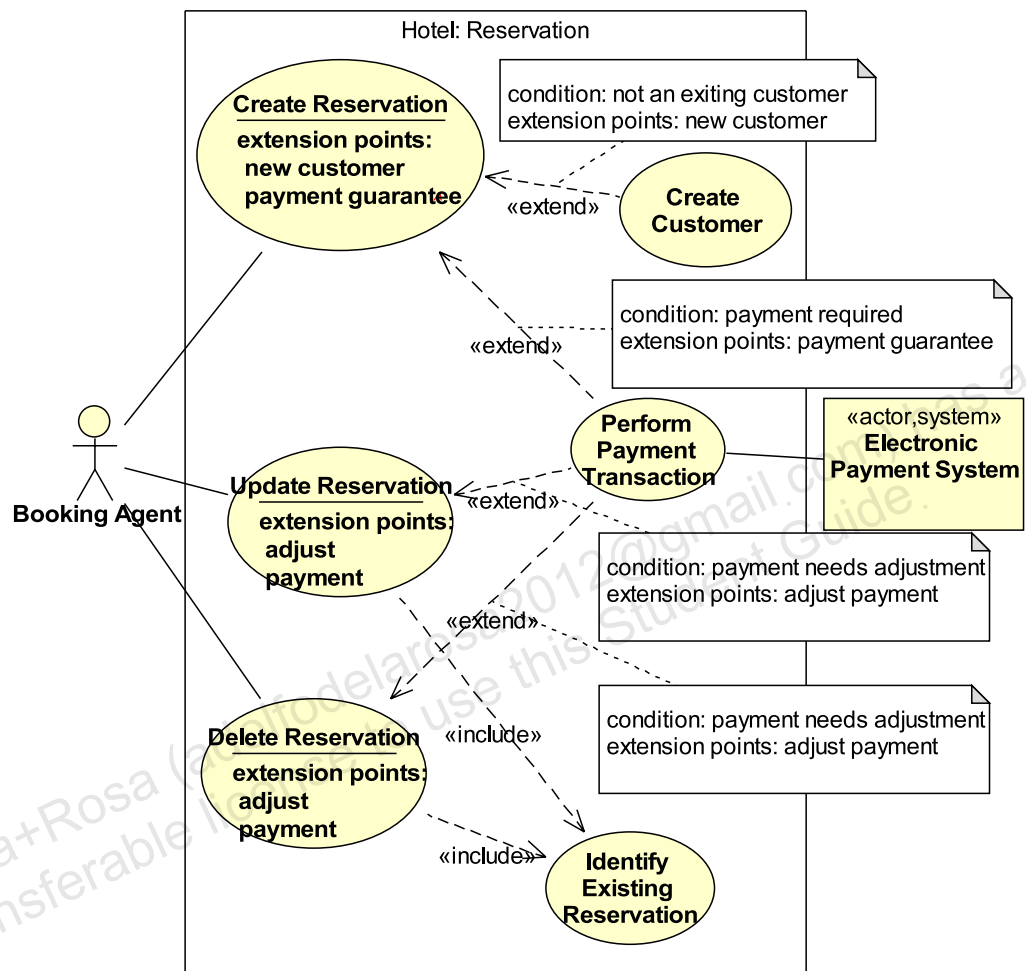
Figure 3-22 shows the use cases and actor relevant to marketing.



**Figure 3-22** Example of Marketing Use Cases and Actor



Figure 3-23 shows some of the use cases and actors relevant to reservations.



**Figure 3-23** Example of Reservation Use Cases and Actors

## Summary

In this module, you were introduced to the UML Use Case diagram and Use Case scenarios. Here are a few important concepts:

- A Use Case diagram provides a visual representation of the big-picture view of the system.
- The Use Case diagram represents the actors that use a system, the use cases that provide a behavior with a definable goal for an actor, and the associations between actors and use cases.
- Use Case diagrams can be elaborated to show a software system based on the goals of the business owner and all the other Stakeholders.
- Use Case diagrams can be elaborated to show use case dependencies by using UML notation for extends, includes, and generalization.
- Complex Use Case diagrams can be broken down into views by using UML packages.