

# Creating Use Case Diagrams

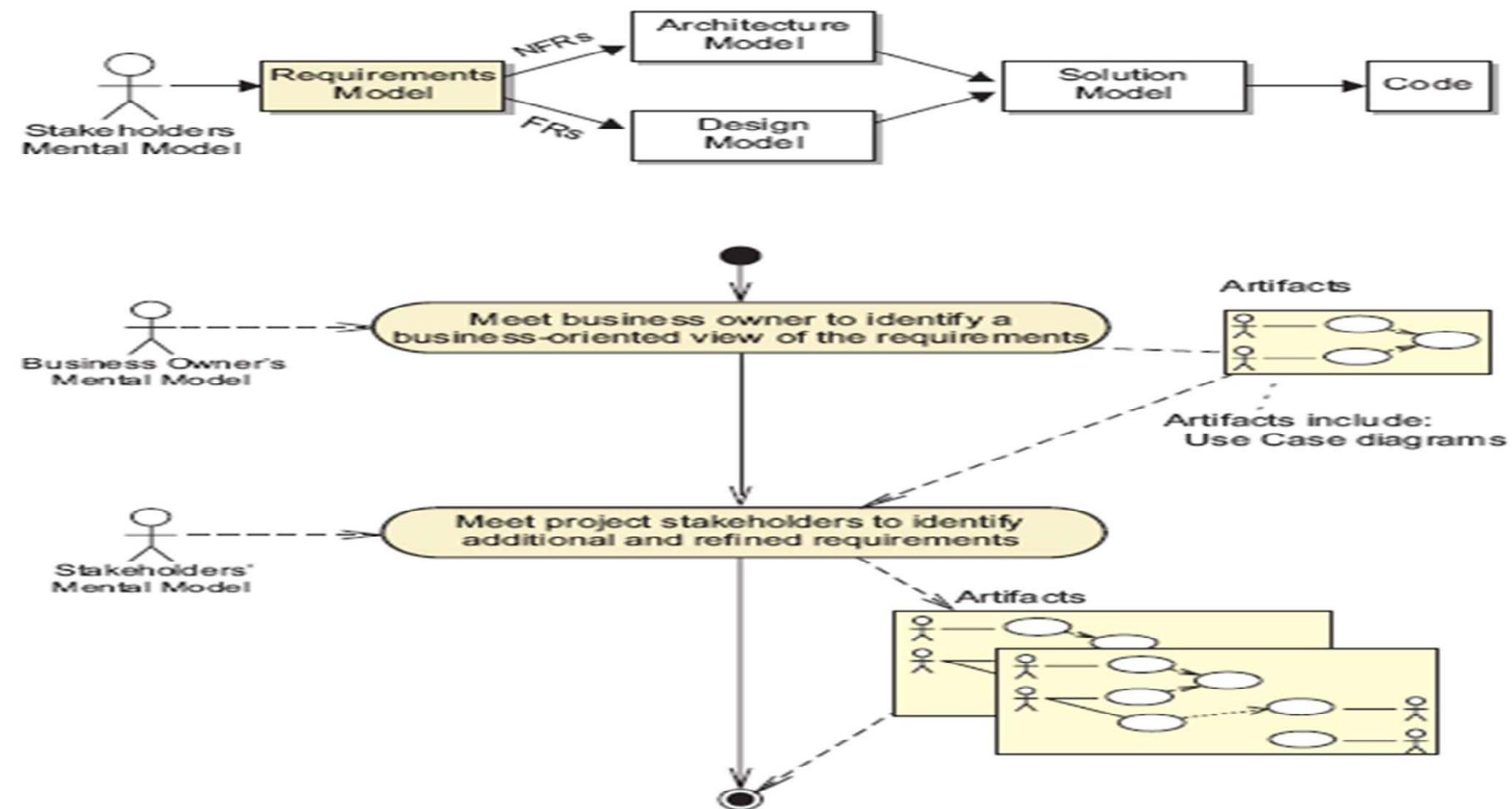
# Objectives

After completing this lesson, you should be able to do the following:

- 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



# Process Map



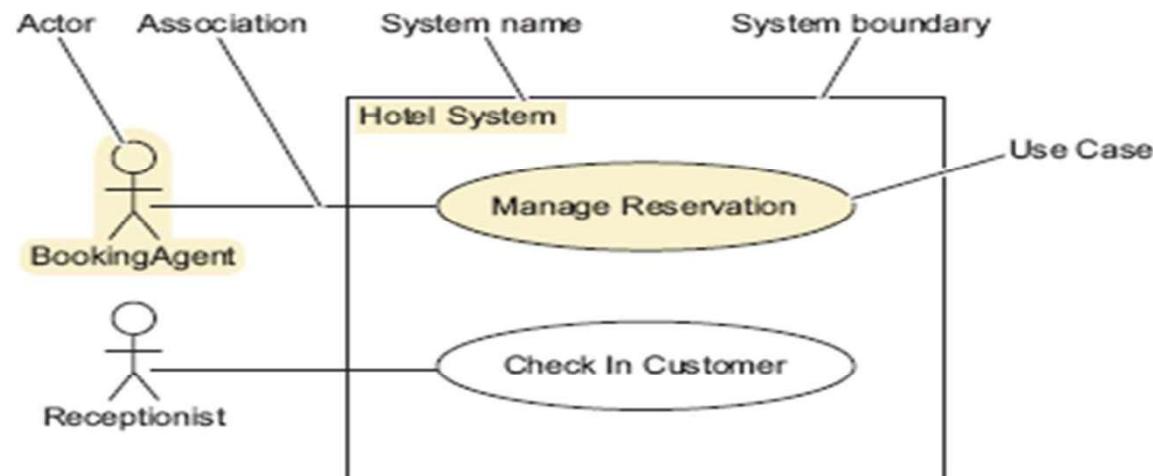
## Justifying the Need for a Use Case Diagram

Following are reasons a Use Case diagram is necessary:

- 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.
- The client-side stakeholders need a big picture view of the system.
- The use cases form the basis from which the detailed FRs are developed.
- Use cases can be prioritized and developed in order of priority.
- Use cases often have minimal dependencies, which enables a degree of independent development.

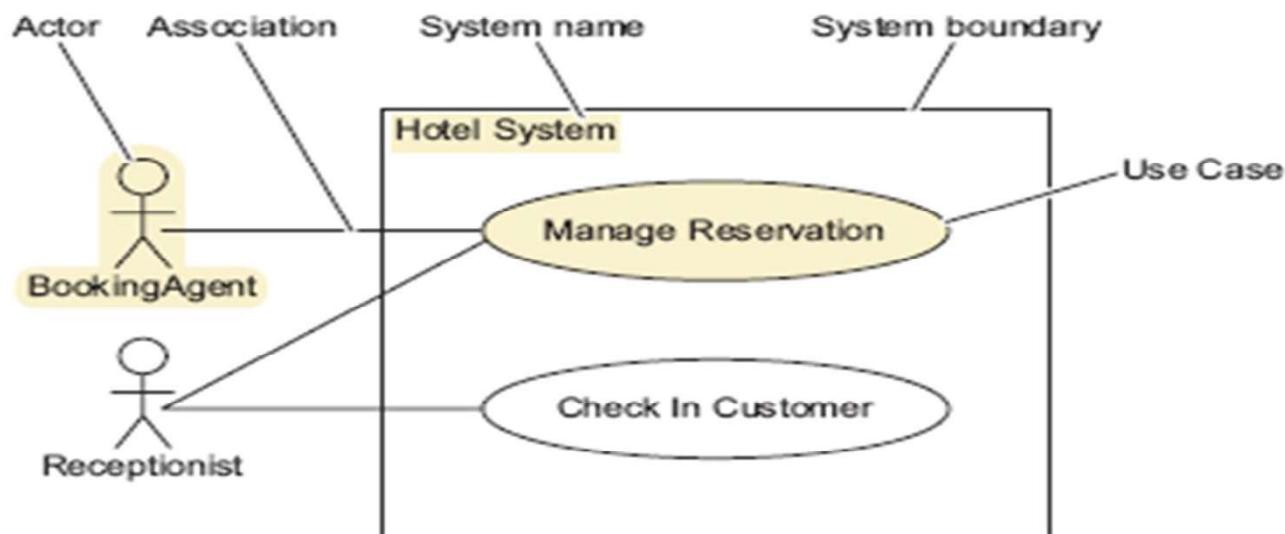
## Identifying the Elements of a Use Case

- A Use Case diagram shows the relationships between actors (roles) and the goals they wish to achieve.
- A physical job title can assume multiple actors (roles).



## Identifying the Elements of a Use Case

- This diagram illustrates an alternate style that explicitly shows an association between the Receptionist actor (role) and the Manage Reservation use case.



An actor:

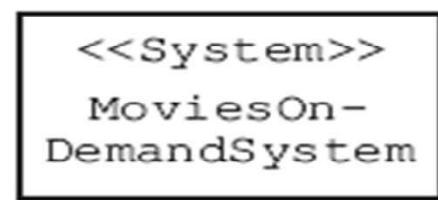
- Models a type of role that is external to the system and interacts with that system
- Can be a human, a device, another system, or time
- Can be primary or secondary
  - Primary: Initiates and controls the whole use case
  - Secondary: Participates only for part of the use case

A single physical instance of a human, a device, or a system may play the role of several different actors.

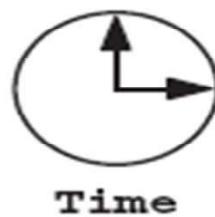
## Actors



This icon represents a human actor (user) of the system.

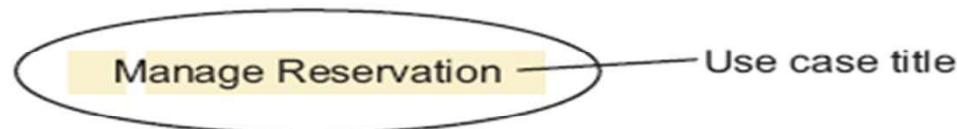


This icon can represent any actor, but is usually used to represent external systems, devices, or time.



This icon represents a time-trigger mechanism that activates a use case.

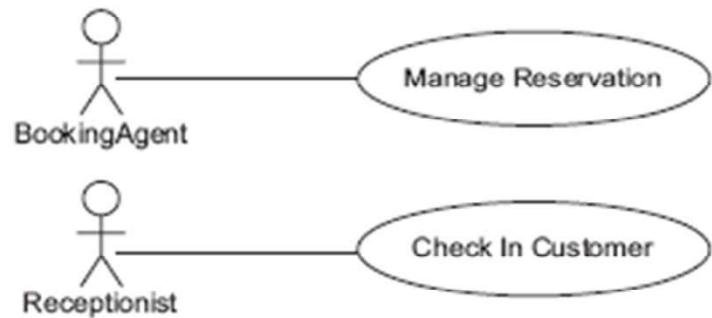
A use case describes an interaction between an actor and the system to achieve a goal.



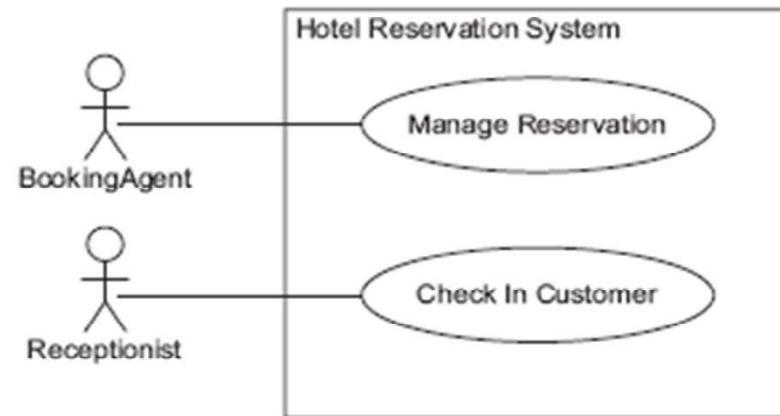
- A use case encapsulates a major piece of system behavior with a definable outcome.
- A use case is represented as an oval with the use case title in the center.
- A good use case title should consist of a brief but unambiguous verb-noun pair.
- A use case can often be UI independent.

## System Boundary

- The use cases may optionally be enclosed by a rectangle that represents the system boundary.



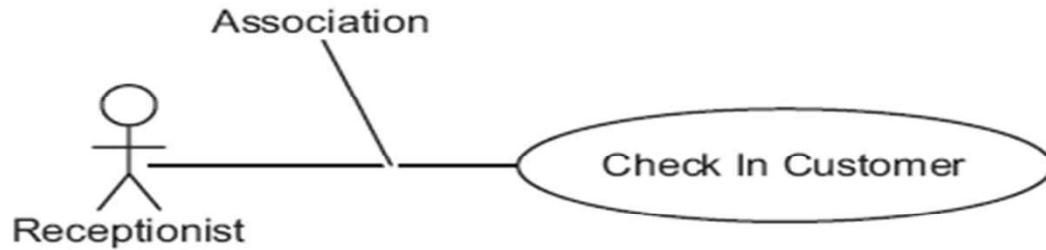
The system boundary box is optional.



This equivalent Use Case diagram shows the system boundary for clarity.

## Use Case Associations

A use case association represents “the participation of an actor in a use case.”



- An actor must be associated with one or more use cases.
- A use case must be associated with one or more actors.
- An association is represented by a solid line with no arrowheads. However, some UML tools use arrows by default.

## 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.
- Alternatively, the diagrams can be created after the meeting from textual notes.

## Use case diagrams

- Use case diagrams present an outside view of the manner the elements in a system behave and how they can be used in the context.

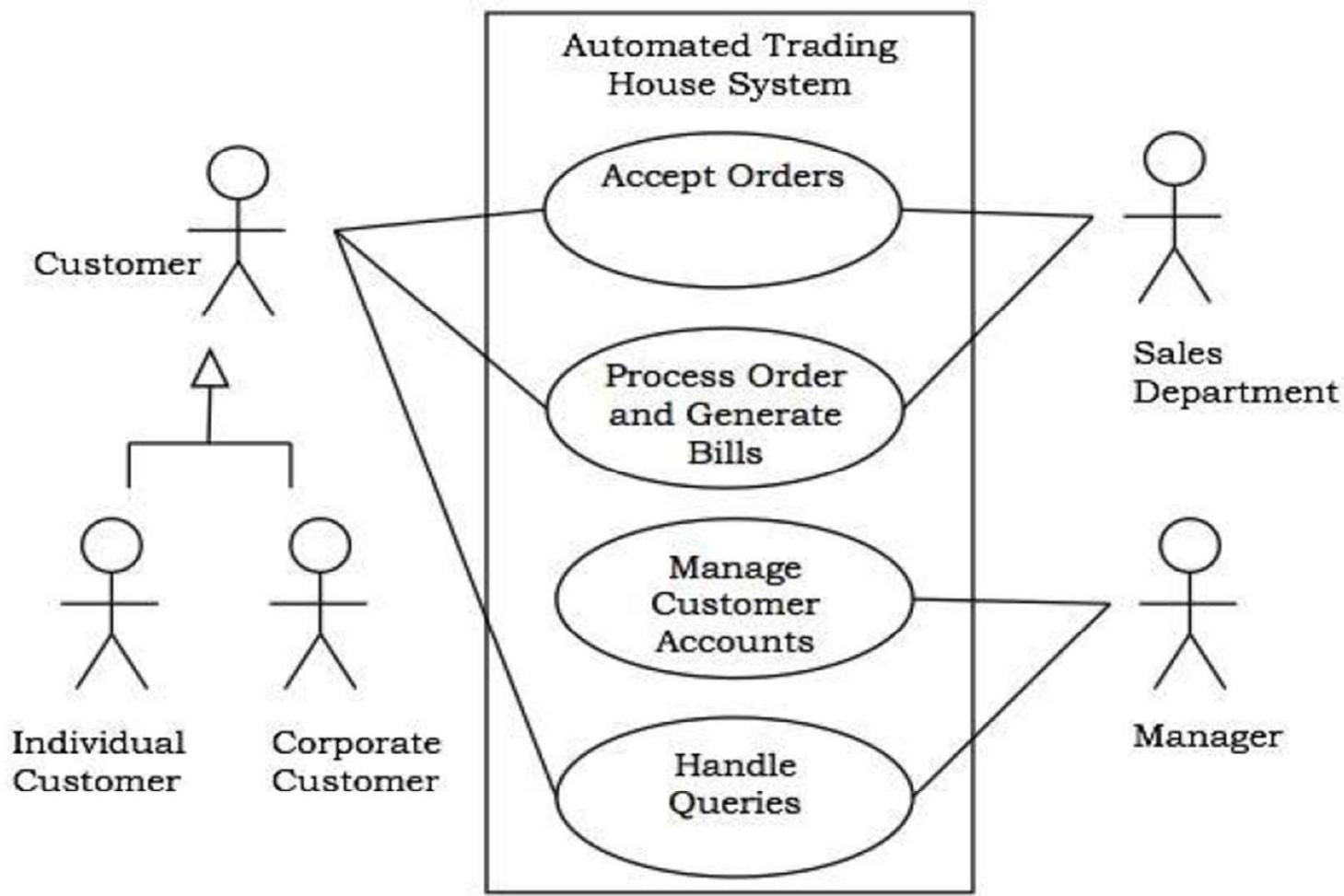
Use case diagrams comprise of:

1. Use cases
2. Actors
3. Relationships like dependency, generalization, and association

Use case diagrams are used:

- To model the context of a system by enclosing all the activities of a system within a rectangle and focusing on the actors outside the system by interacting with it.
- To model the requirements of a system from the outside point of view.

## Example : 1



## Creating the Initial Use Case Diagram

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.

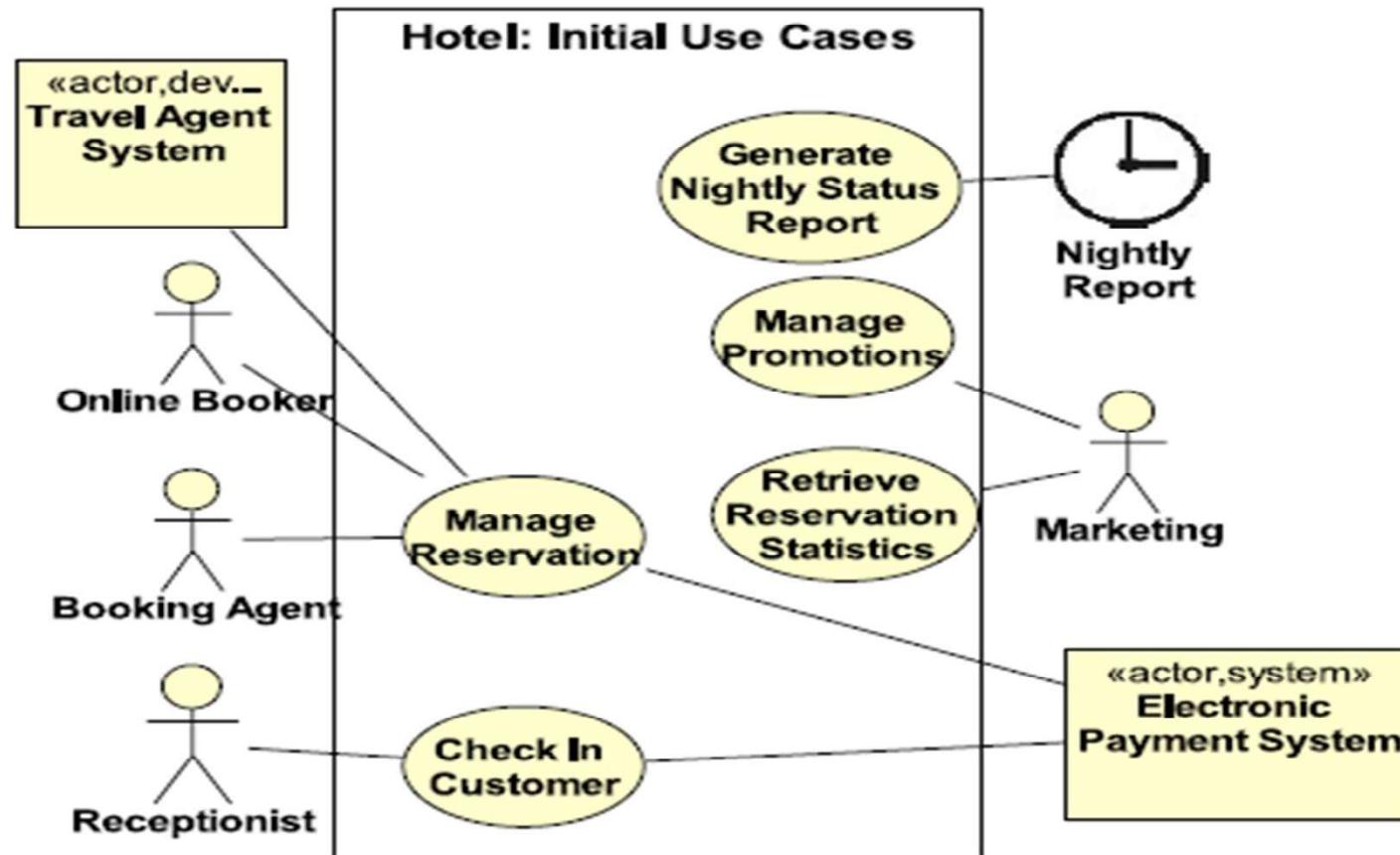
A reservation can also be made electronically from the Travel Agent system and also by customers directly via the internet.

## Creating the Initial Use Case Diagram

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

## Example: 2 Creating the Initial Use Case Diagram



## Identifying additional Use Cases

- During the meeting with the business owner, you will typically discover 10 to 20 percent of the use cases 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

## Identifying additional Use Cases

The time of discovery 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.

## Identifying additional Use Cases

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.

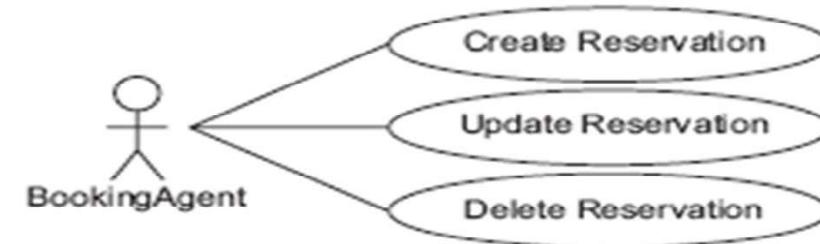
## Use Case Elaboration

- 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. In this case, you can introduce new use cases that separate the workflows.

Example:



Becomes:

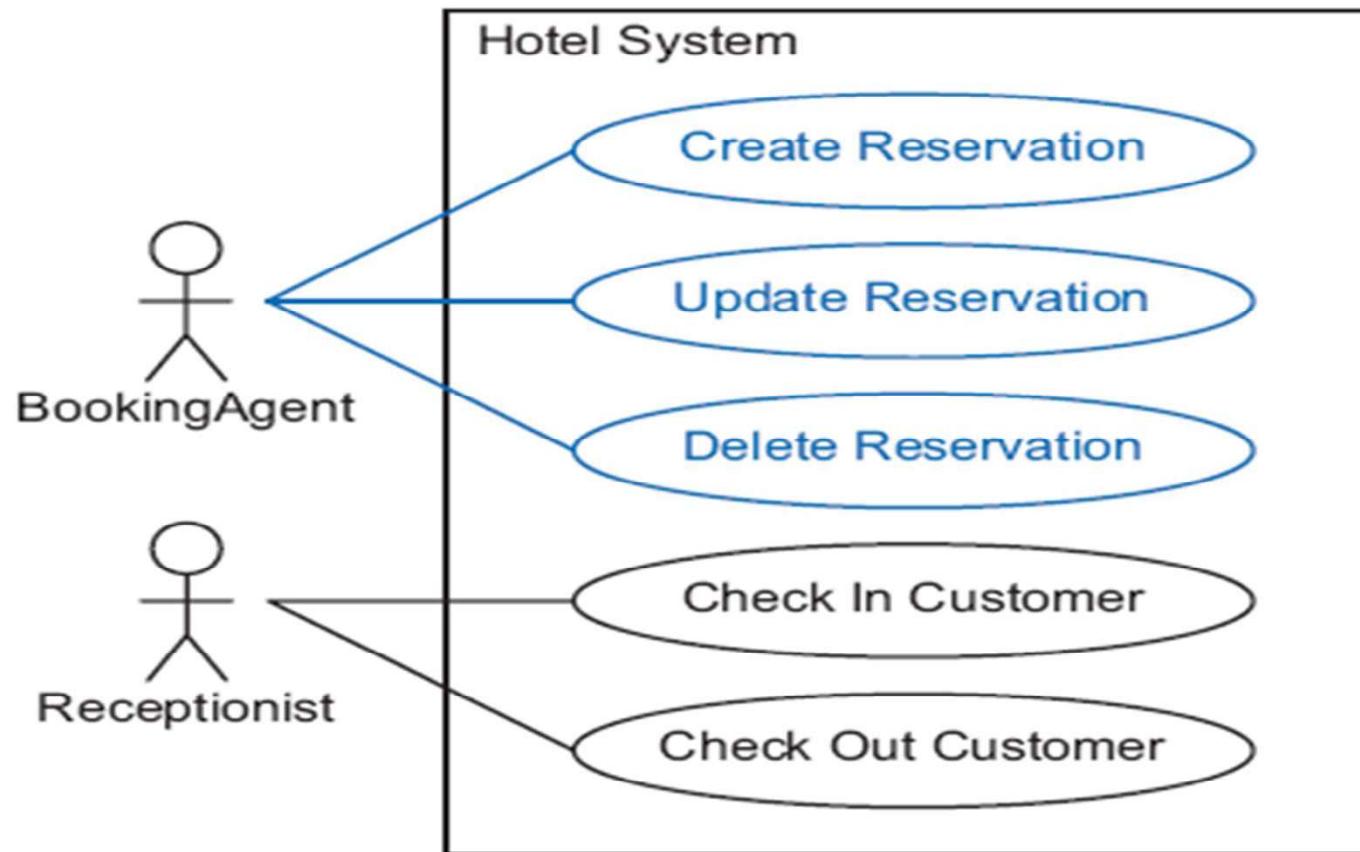


## Expanding High-Level Use Cases

- 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.
- If several scenarios have a different starting point, these scenarios might represent different use cases.

## Expanding High-Level Use Cases

- The expanded diagram:



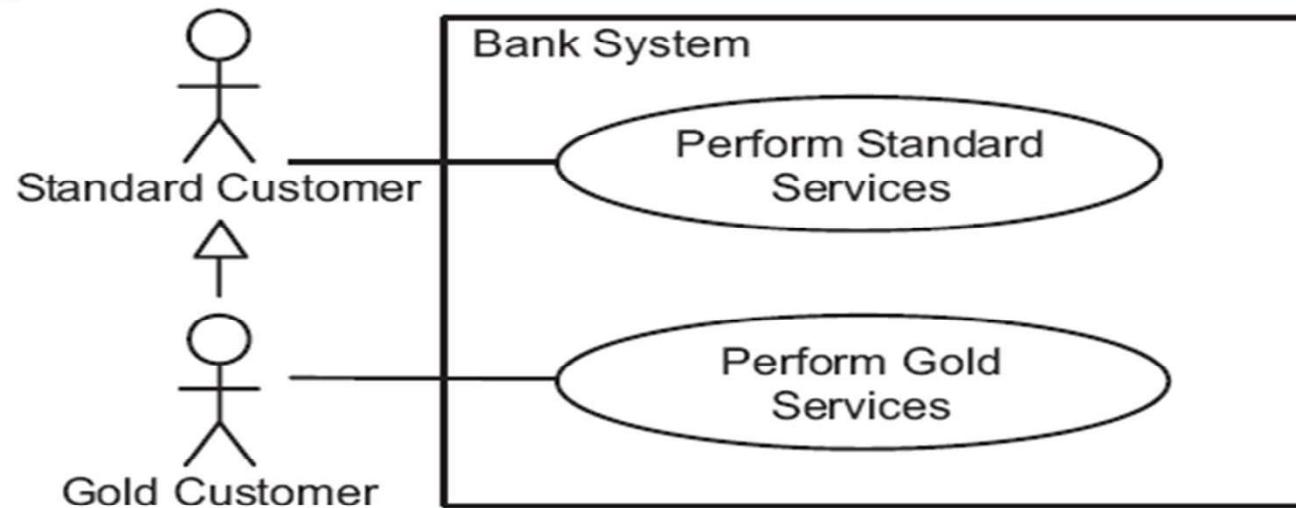
## Analyzing Inheritance Patterns

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

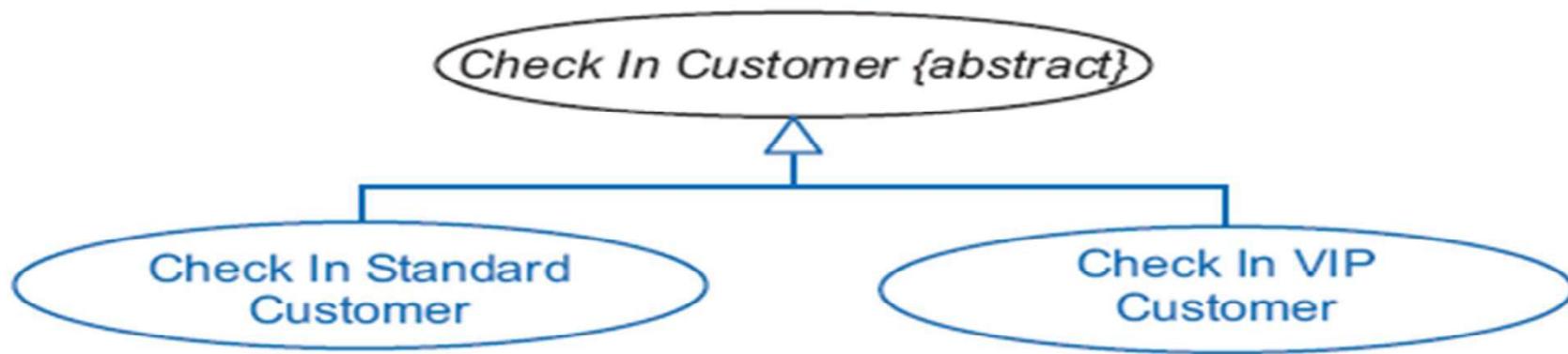
- An actor can inherit all of the use case associations from the parent actor.



- This inheritance should be used only if you can apply the “*is a kind of* rule between the actors.

## Use Case Specialization

- A use case can be *subclassed into multiple, specialized use cases*:



- Use case specializations are *usually identified by significant variations in the use case scenarios*.
- If the base use case cannot be instantiated, you must mark it as abstract.

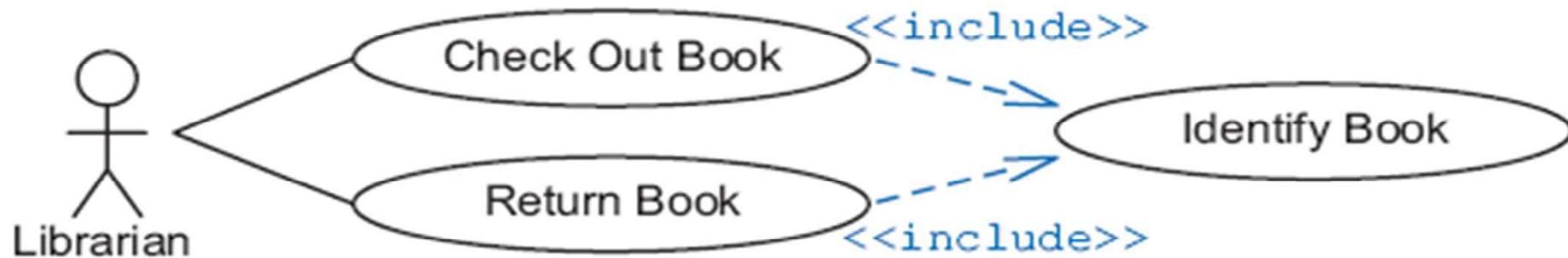
## Analyzing Use Case Dependencies

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

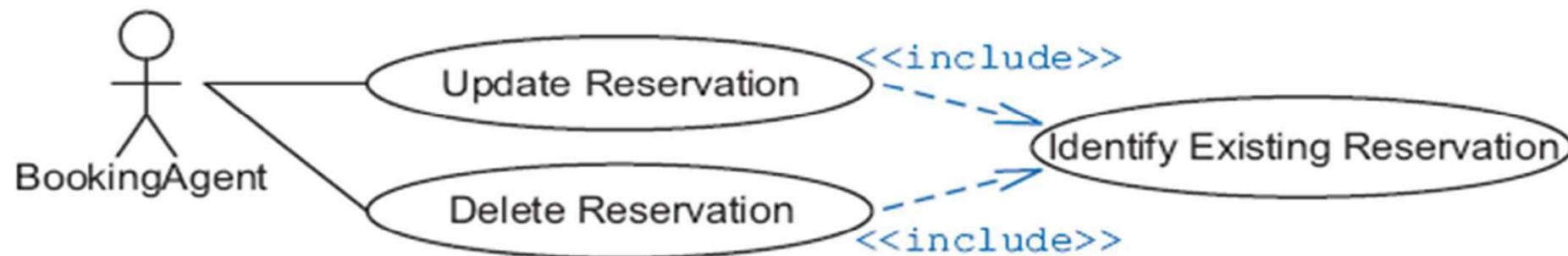
- The include dependency enables you to identify behaviors of the system that are common to multiple use cases.
- This dependency is drawn like this:



## The «include» Dependency

Identifying and recording common behavior:

- Review the use case scenarios for common behaviors.
- Give this behavior a name and place it in the Use Case diagram with an «include» dependency.



## The «include» Dependency

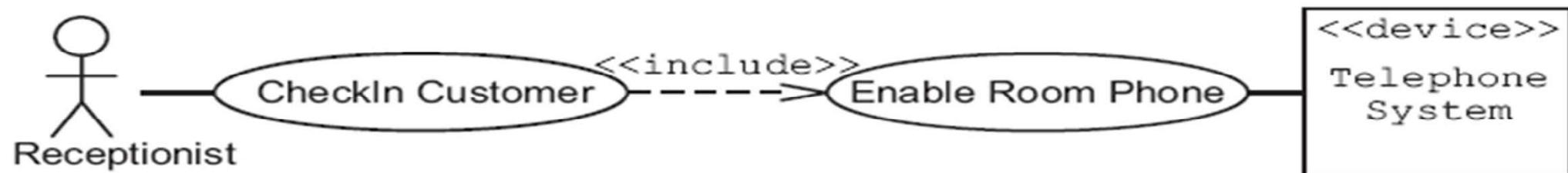
Identifying behavior associated with a secondary actor:

- Review the use case scenarios for significant behavior that involves a secondary actor.



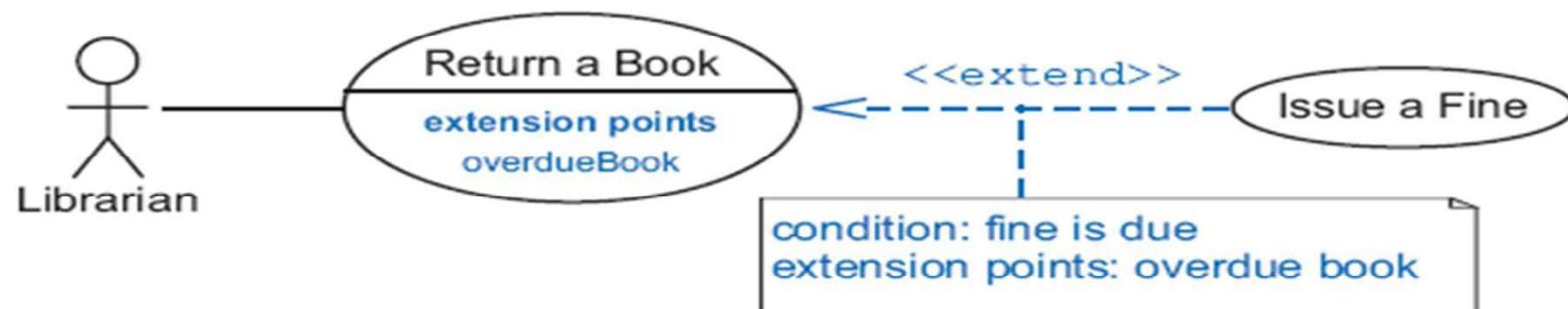
## The «include» Dependency

- Split the behavior that interacts with this secondary actor. Give this behavior a Use Case title, and place it in the Use Case diagram with an «include» dependency.



## The «extend» Dependency

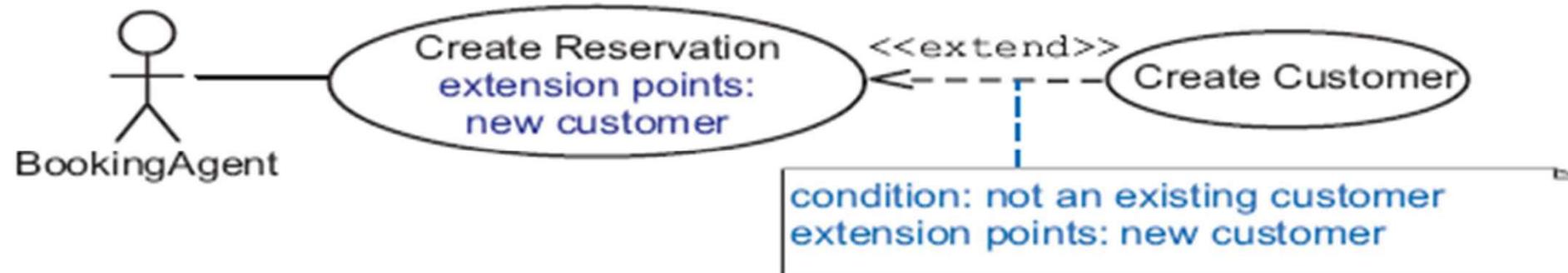
- The extend dependency enables you to identify behaviors of the system that are not part of the primary flow, but exist in alternate scenarios.
- This dependency is drawn like this:



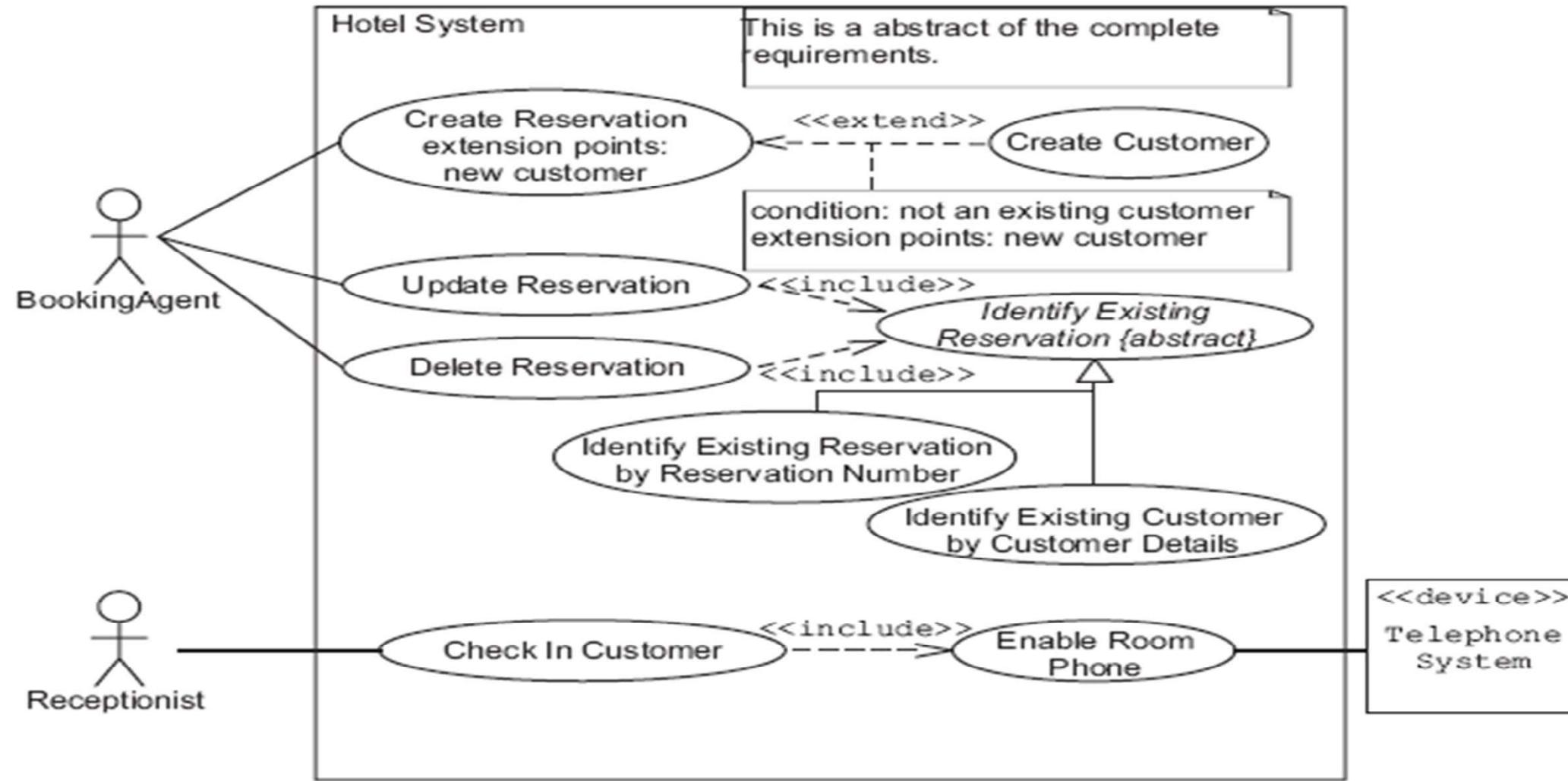
## The «extend» Dependency

Identifying and recording behaviors associated with an alternate flow of a use case:

- Review the use case scenarios for significant and cohesive sequences of behavior.
- Give this behavior a name and place it in the Use Case diagram with a «extend» dependency.

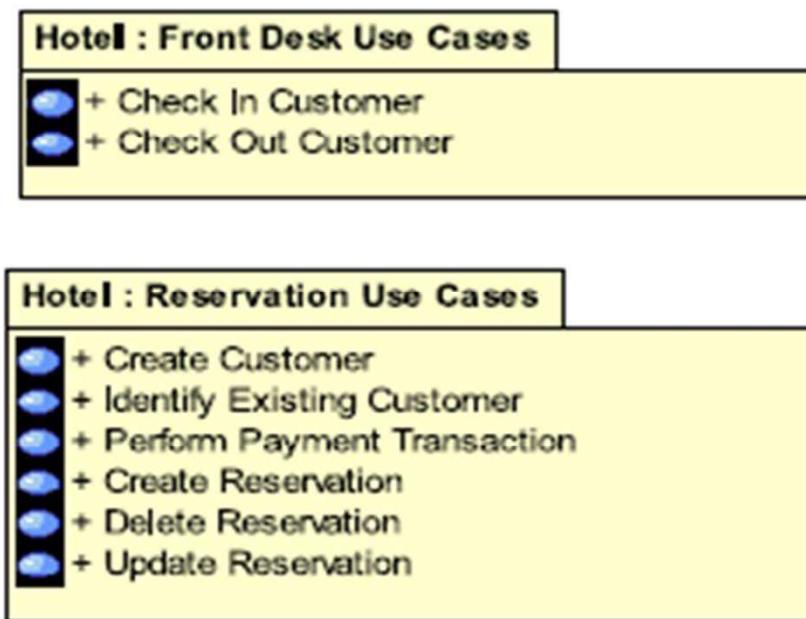


## A Combined Example



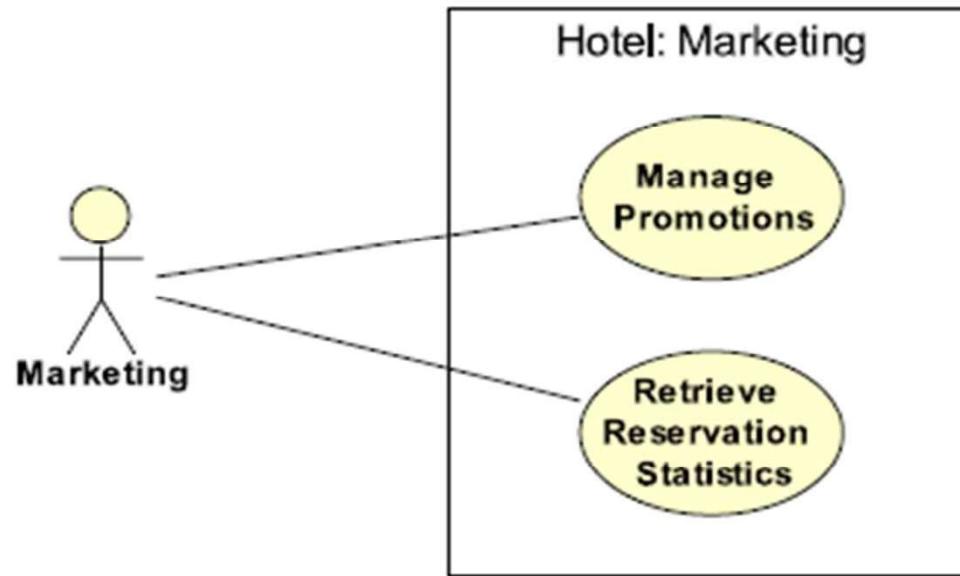
## Packaging the Use Case Views

- 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 packages.

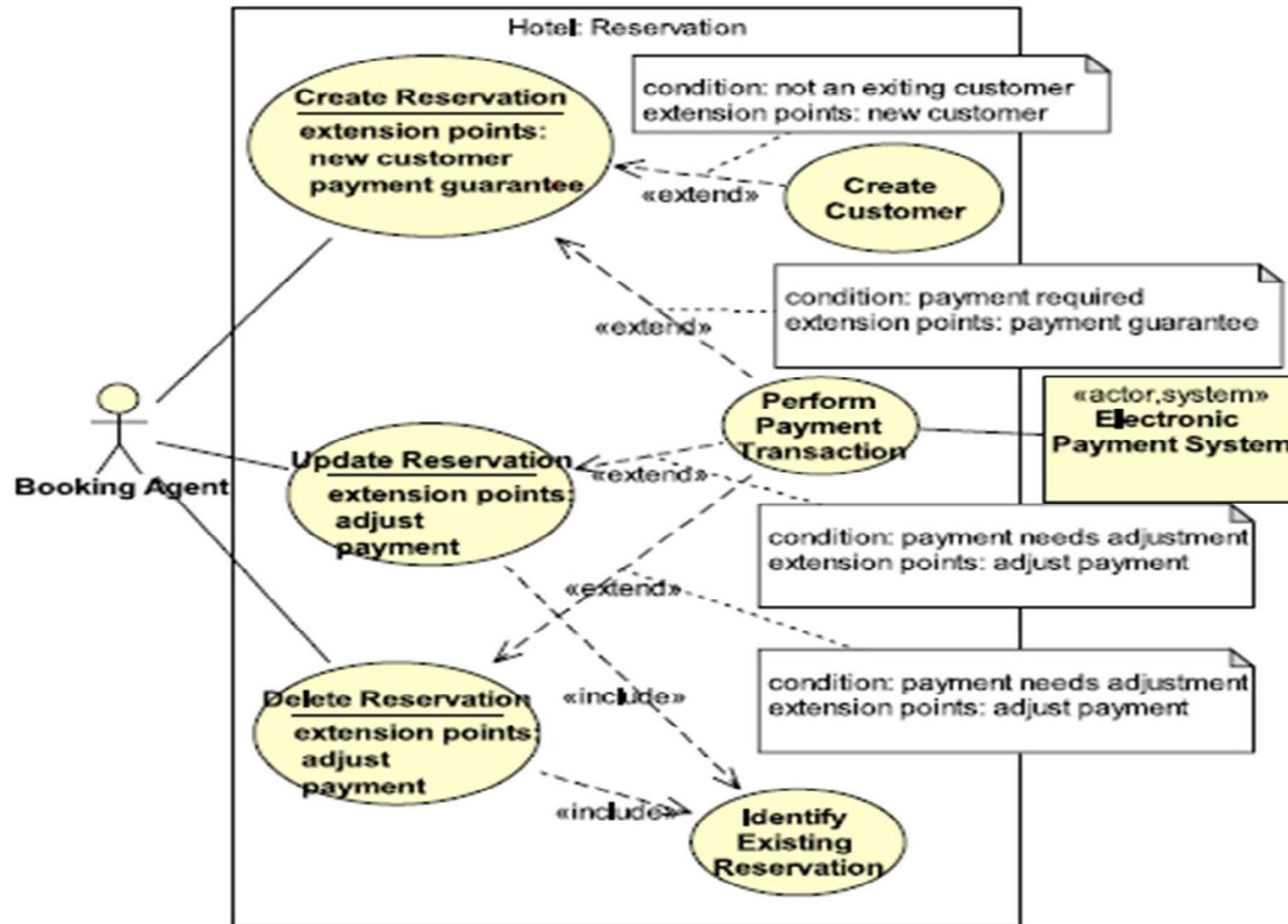


## Packaging the Use Case Views

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



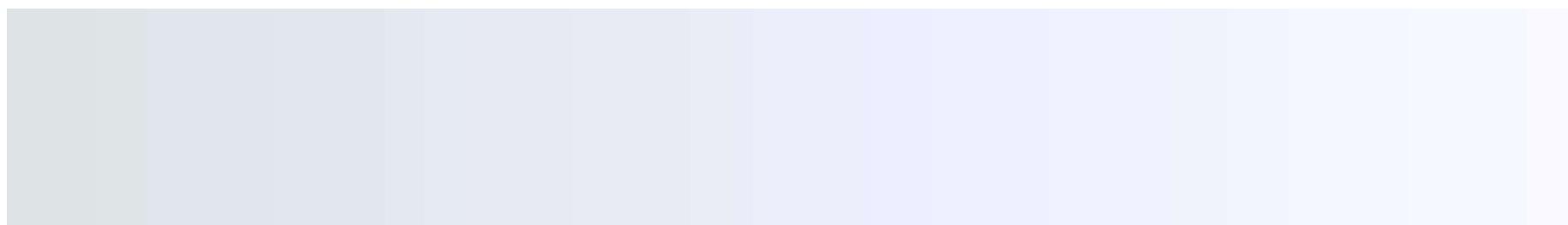
# Packaging the Use Case Views



## Summary

In this lesson, you should have learned the following:

- 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



## Practice : Overview

This practice covers the following topics:

- Working with Use Case Diagrams

