Use Case Diagrams

Divide the requirements into Use cases

➔ You can then design & implement each use case

Use Case Diagrams

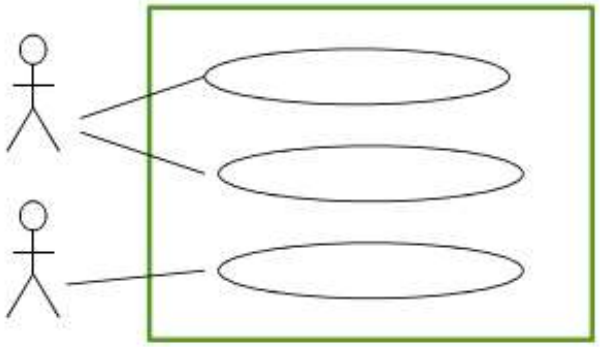
- Systems interact with their external environment.
- The set of *external objects* and their *interactions* with the system form the basis for the requirements analysis of the system.

Use Case Diagrams

- Consist of
  - Named pieces of capabilities: *use cases*
  - Persons or things invoking capabilities: *actors*
  - Relationships between actors and use cases: *associations*
  - Non functional requirements: *constraints*

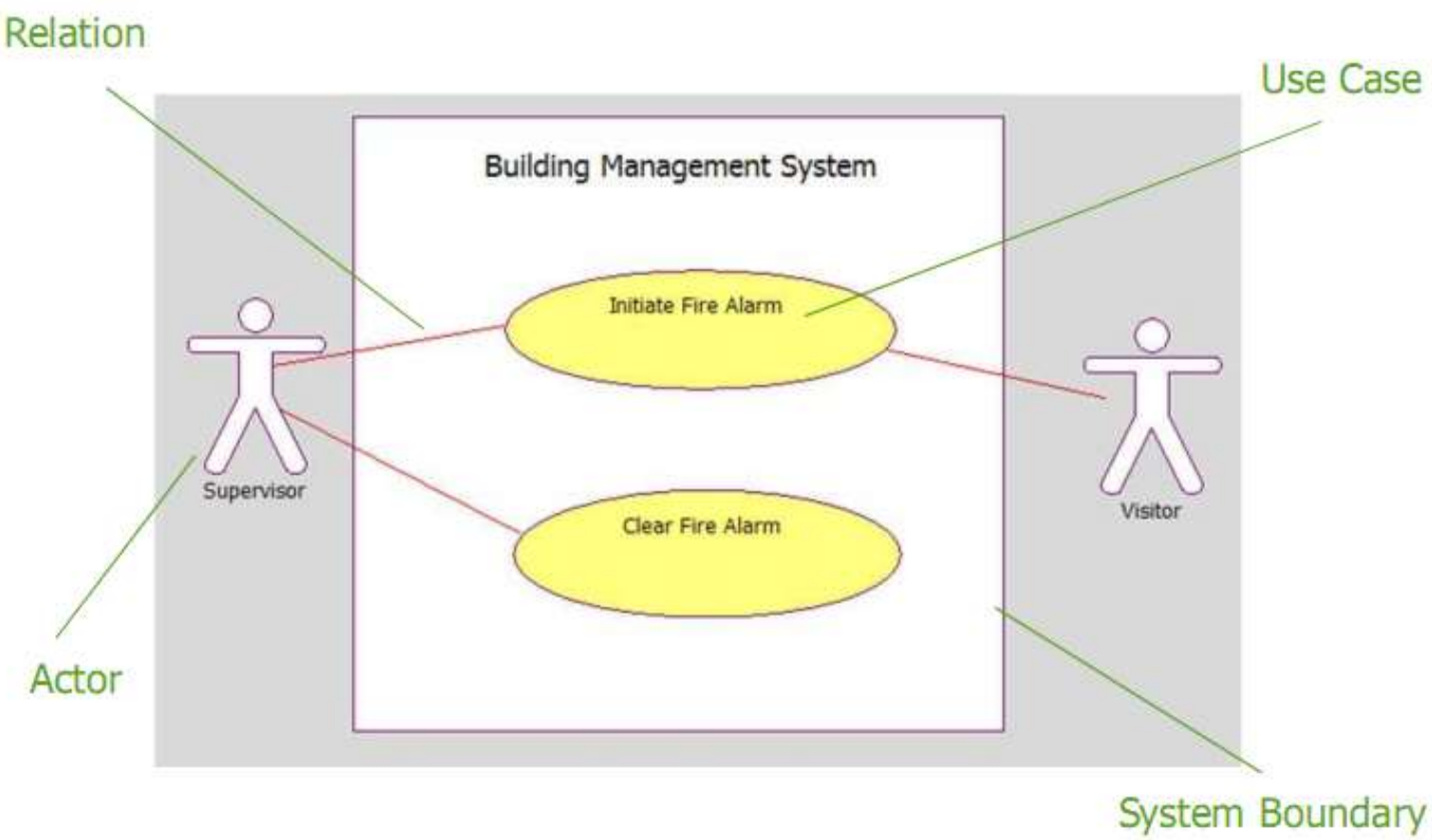
Use Case Diagrams

- In order to define a use case diagram, we should:
  - Find the system boundary
  - Find actors
  - Find use cases



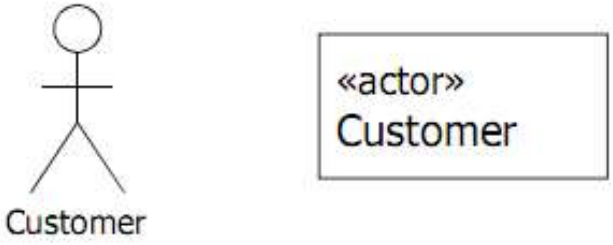
- Use Case Diagrams allow us to identify the *system boundary*, *who* or *what* uses the system, and what *capabilities* the system should offer

Basic Syntax



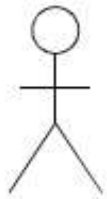
What are actors?

- An actor is anything that interacts *directly* with the system
  - Actors identify *who* or *what* uses the system and so indicate where the system boundary lies
- Actors are *external* to the system
- An Actor specifies a *role* that some external entity adopts when interacting with the system



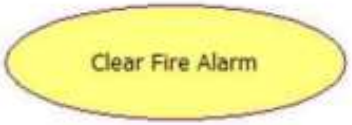
Identifying Actors

- When identifying actors ask:
  - Who or what uses the system?
  - What roles do they play in the interaction?
  - What other systems use this system?
  - Who gets and provides information to the system?



What are use cases?

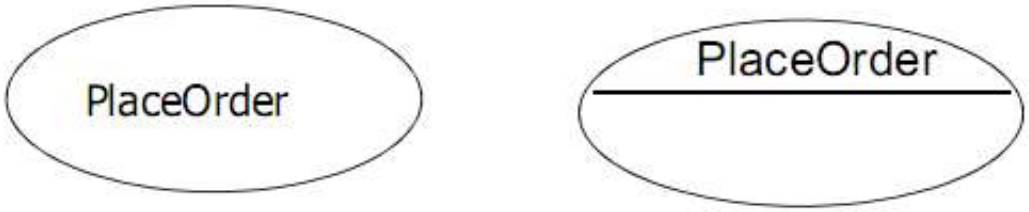
- A use case is something an actor needs the system to do. It is a "**case of use**" of the system by a specific actor
- Is a named ***capability*** of a system



- Describes a ***complete course of events***
- Describes why the user interacts with the system
- Returns a result visible to one or more actors
- Does not reveal or imply internal structure of the system
- Can be used to organize requirements
- Can be used to help with project planning

What are Use Cases?

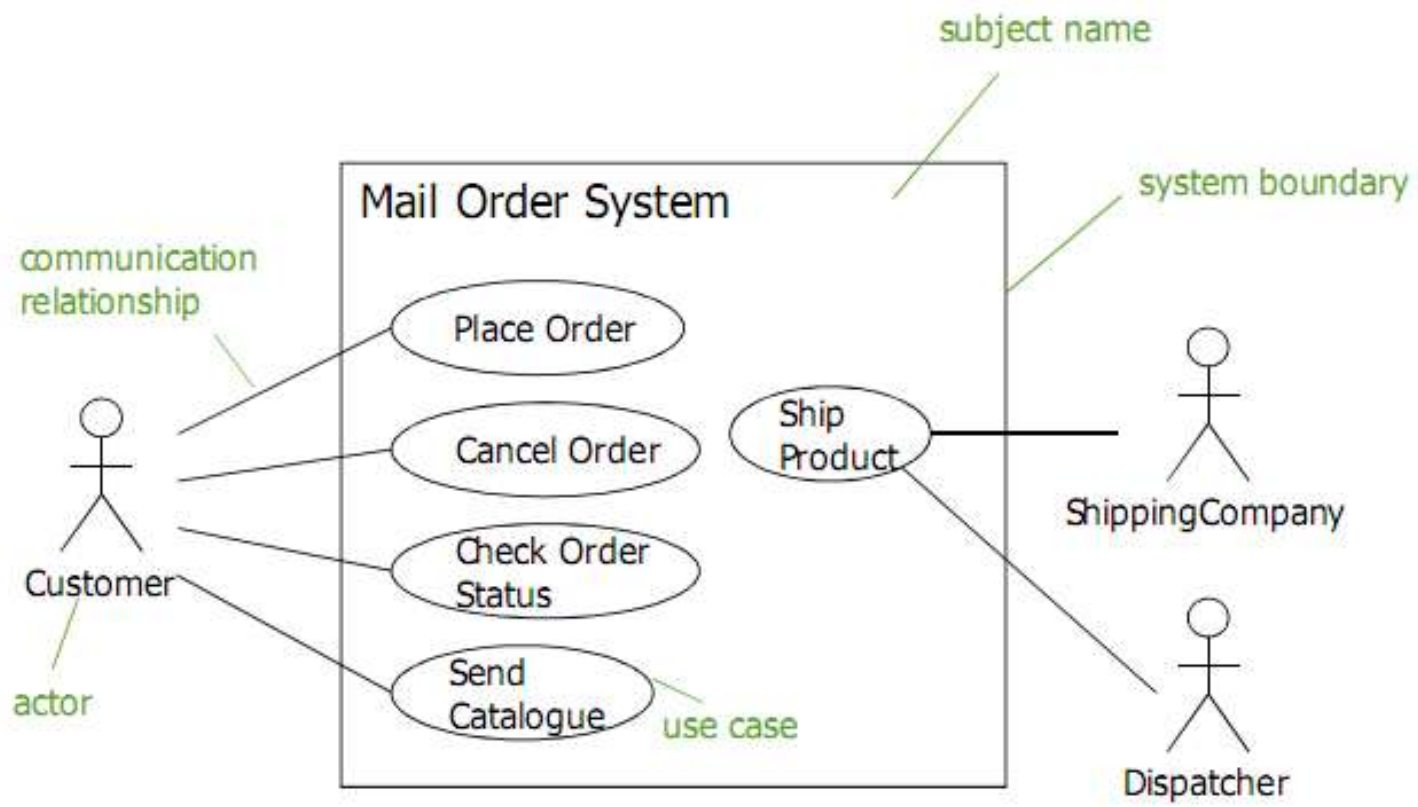
- Use cases are started by an actor or the system
  - The *primary actor* triggers the use case
  - Zero or more *secondary actors* interact with the use case in some way
- Use cases are *always* written from the point of view of the actors



Identifying use cases

- Start with the list of actors that interact with the system
- When identifying use cases ask:
  - What functions will a specific actor want from the system?
  - Does the system store and retrieve information? If so, which actors trigger this behaviour?
  - What happens when the system changes state (e.g., system start and stop)? Are any actors notified?
  - Are there any external events that affect the system? What notifies the system about those events?
  - Does the system interact with any external system?
  - Does the system generate any reports?

Use Case Diagram



Use Case – Course of Events

- The flow of events lists the steps in a use case
- The main flow is always the *happy day* or *perfect world* scenario
  - Everything goes as expected and desired, and there are no errors, deviations, interrupts, or branches
  - Alternatives can be shown by branching or by listing under Alternative flows

- It *usually* begins by an actor doing something
  - A good way to start a flow of events is:
    1) The use case starts when an <actor> <function>
- The flow of events should be a sequence of short steps that are:
  - Declarative
  - Numbered,
  - Time ordered

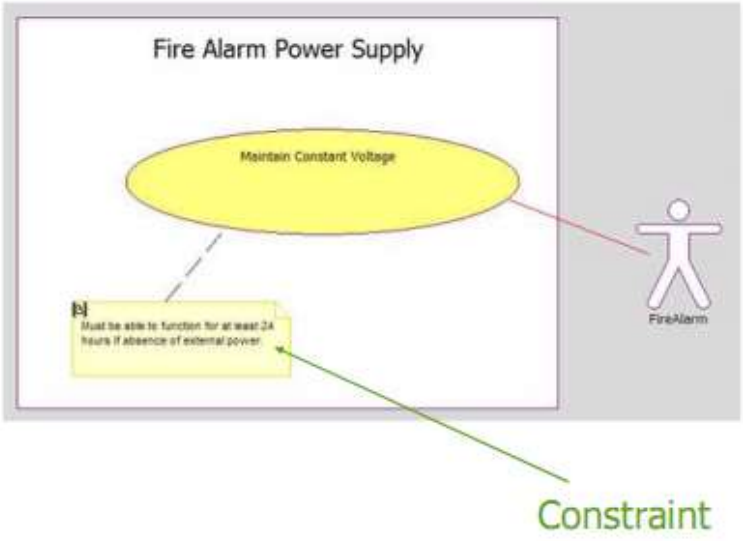
Use Case Description

- Various format templates are available for detailed use cases.

use case name	Use case: Place Order
use case identifier	ID: 1
brief description	Brief description: The customer places an order through the Mail Order System
the actors involved in the use case	Primary actors: Customer
	Secondary actors: None
the system state before the use case can begin	Preconditions: 1. The customer must have an account
the actual steps of the use case	Main flow: 1. The customer logs in 2. The customer browses the list of items 3. The customer selects an item and places it in the cart ... n. A confirmation is sent to the customer
the system state when the use case has finished	Postconditions: 1. The order is logged 2. The customers received a confirmation

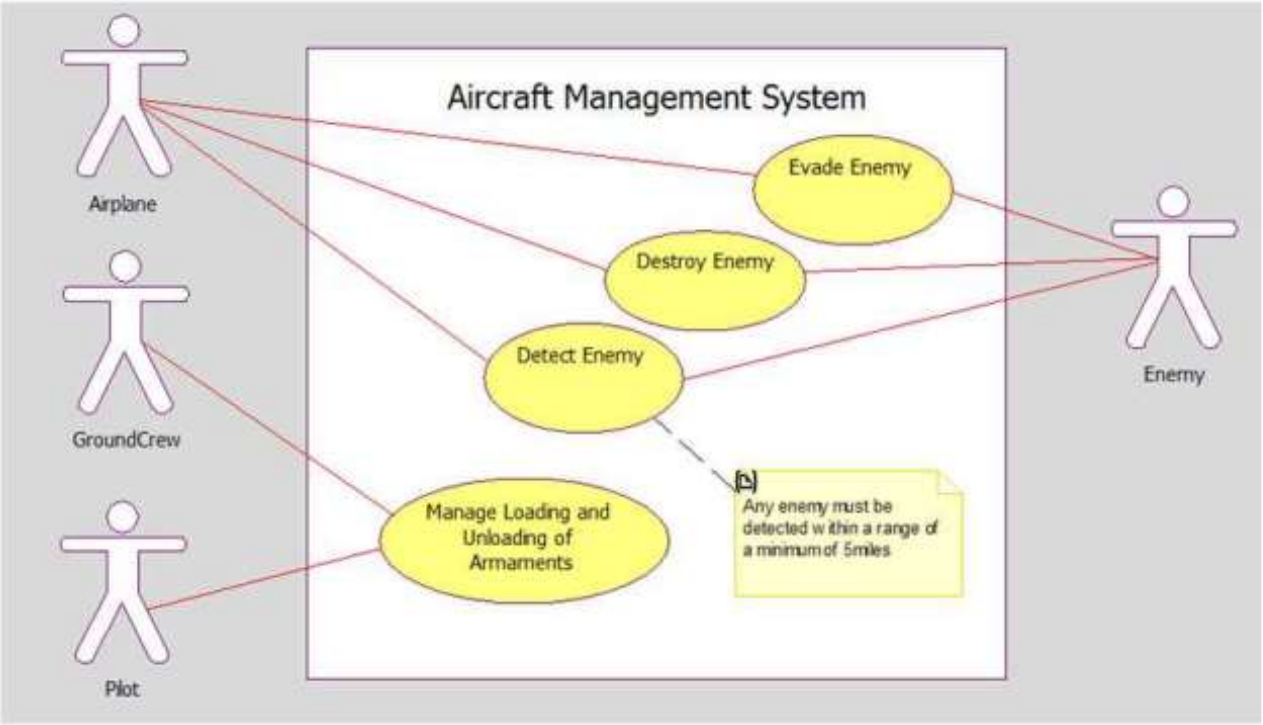
Use Case Constraints

- A ***constraint*** is a way to capture how well the system must perform: *Non Functional Requirements* or *Quality of Service*.
- It can be used to express:
  - Worst-case execution time
  - Average execution time
  - Throughput
  - Predictability
  - Capacity
  - Safety
  - Reliability



An Example

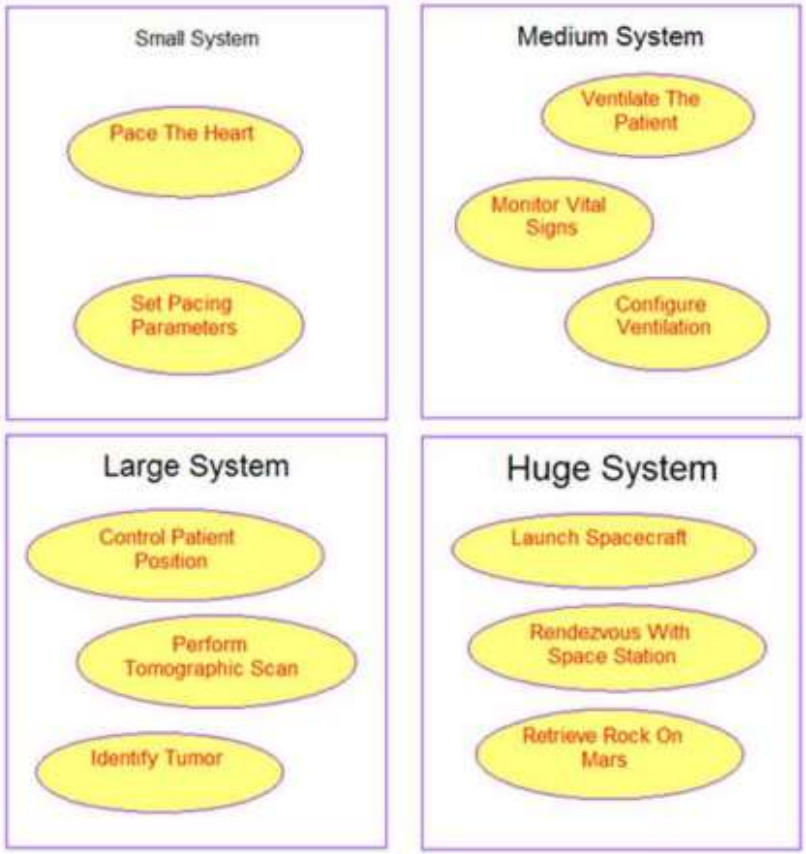
- Note what is shown: Actors / Use cases / QoS requirements



- Note what *is not shown*: Internal workings of the actors or system

Examples of Use Cases

- As the System gets bigger, the use cases get more abstract and “high level”



Use Cases Are Not …

Use Cases are *NOT*

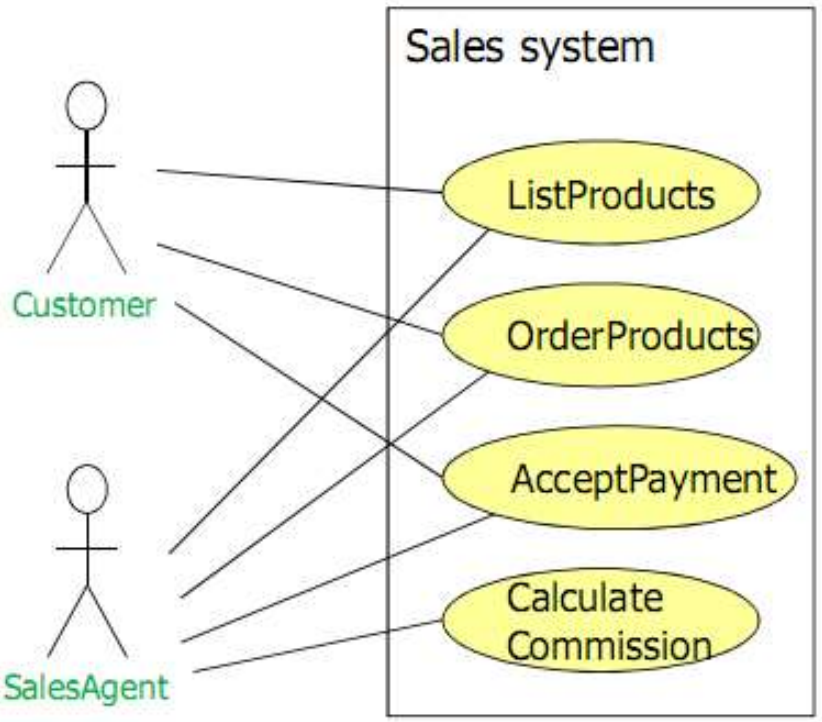
- A functional decomposition model
- They do not capture anything the actors do outside of the System

Outline

- Dynamic Aspects
- Use Case Diagrams
  - Basic Components
  - Advanced Use Case Modeling
    - Actor generalization
    - Use Case generalization
    - <<include>> - between use cases
    - <<extend>> - between use cases
- Activity Diagrams
- Statechart Diagrams
- Interaction Diagrams

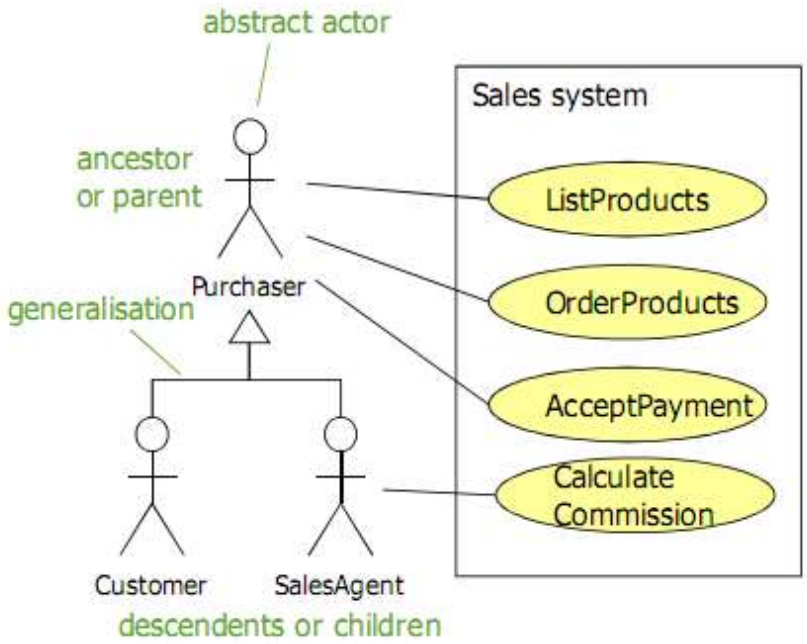
Actor Generalization - example

- The *Customer* and the *Sales Agent* actors are very similar
- They both interact with *List products*, *Order products*, *Accept payment*
- Additionally, the *Sales Agent* interacts with *Calculate commission*
- Our diagram is a *mess* – can we simplify it?



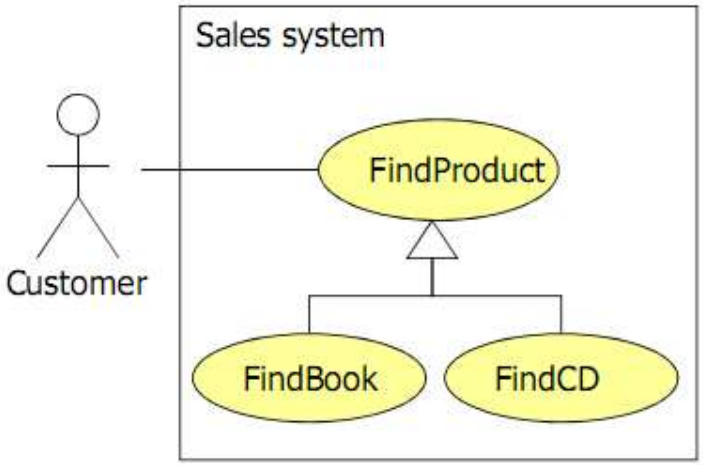
Actor generalization

- If two actors communicate with the same set of use cases in the same way, then we can express this as a generalisation to another (possibly abstract) actor
- The descendent actors inherit the roles and relationships to use cases held by the ancestor actor



Use case generalization

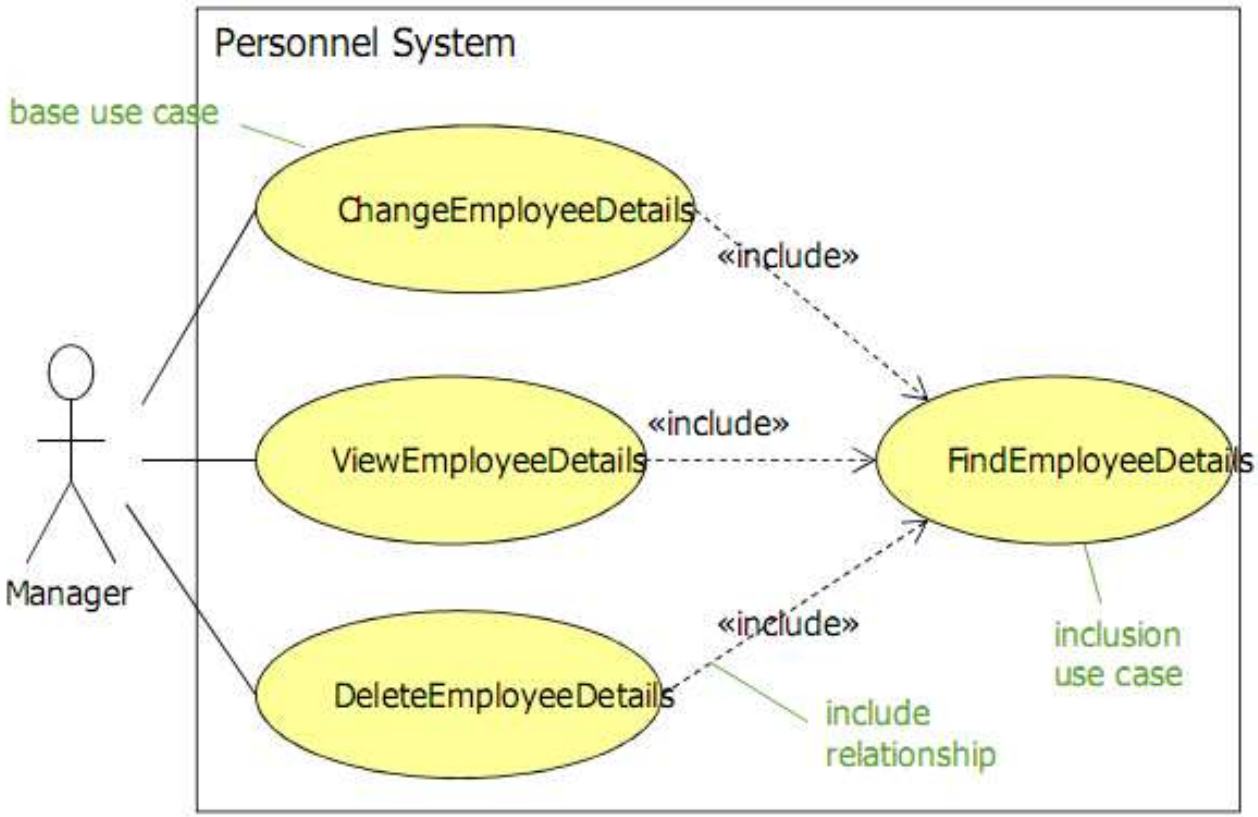
- Use case generalization is used to express some high-level functional need of a system without going into specifics.
- The ancestor use case must be a more general case of one or more descendant use cases
- Child use cases are more specific forms of their parent
- They can inherit, add and override features of their parent



Use case Inclusion

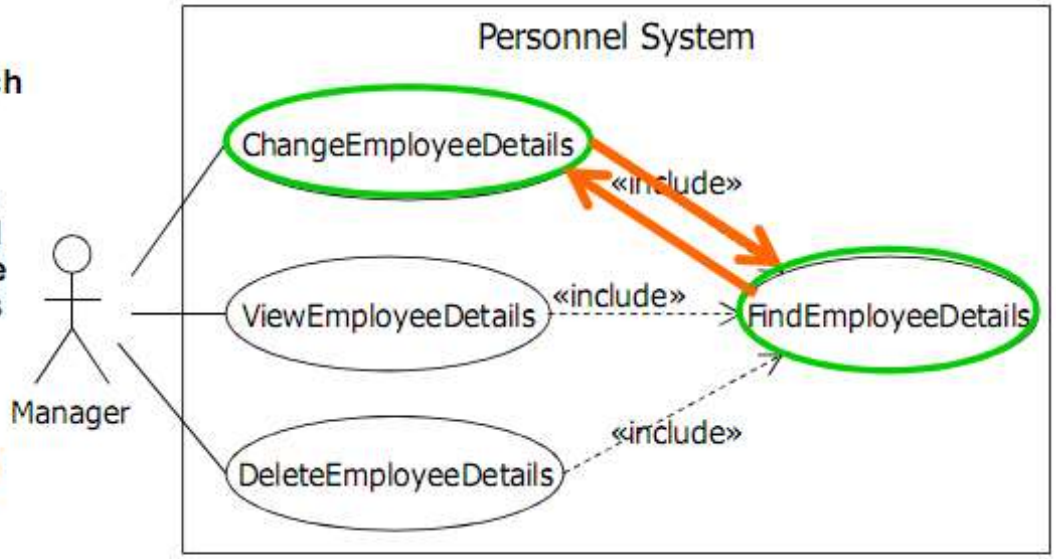
- When use cases share common behaviour we can factor this out into a separate inclusion use case and «include» it in base use cases
- The base use case executes until the point of inclusion: control passes to the inclusion use case which executes
  - When the inclusion use case is finished, control passes back to the base use case which finishes execution
- Note:
  - Base use cases are *not complete* without the included use cases
  - Inclusion use cases may be complete use cases, or they may just specify a fragment of behaviour for inclusion elsewhere

Use case Inclusion

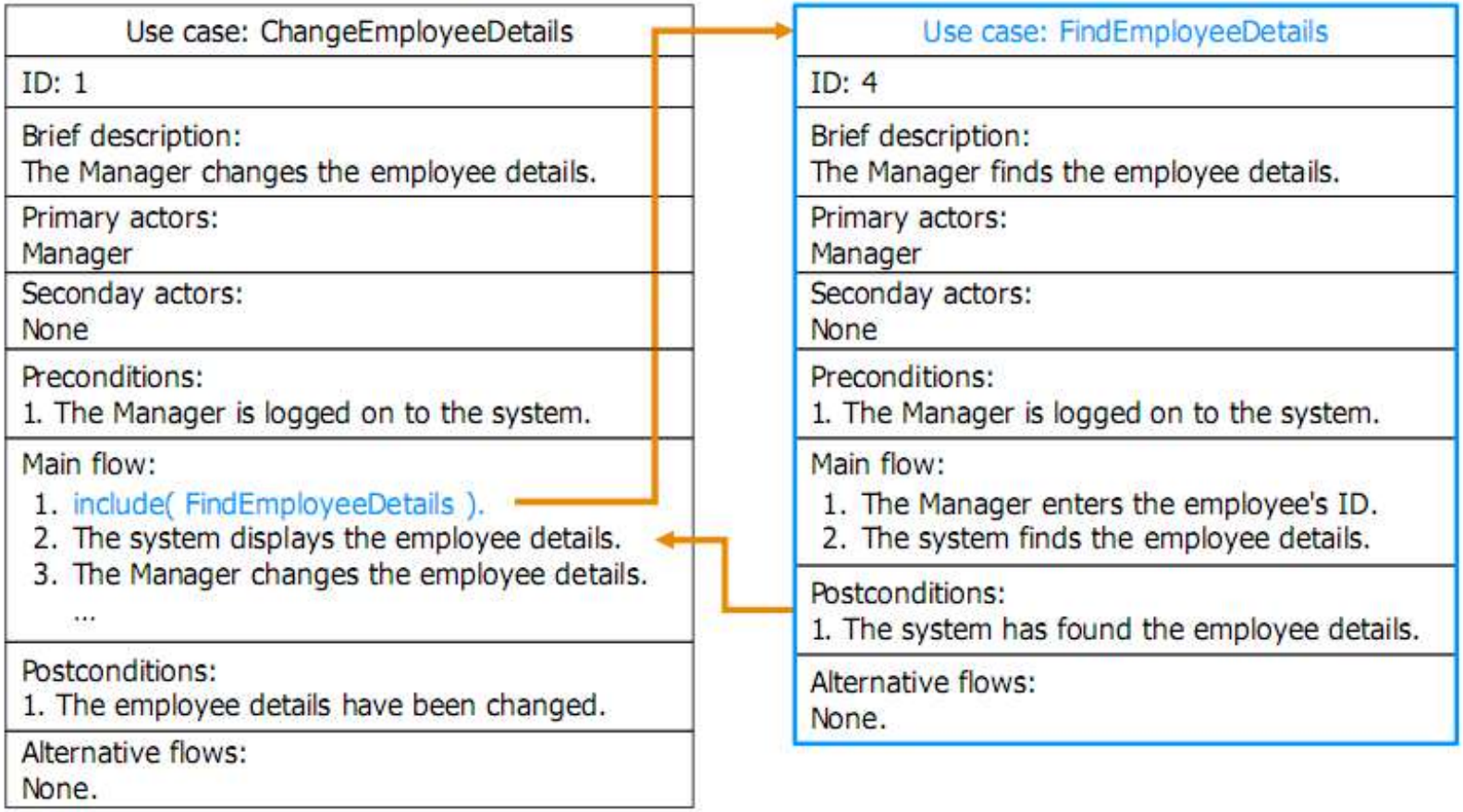


Use Case -- <<include>>

- The base use case executes until the point of inclusion: include(InclusionUseCase)
  - Control passes to the inclusion use case which executes
  - When the inclusion use case is finished, control passes back to the base use case which finishes execution
- Note:
  - Base use cases are *not complete* without the included use cases
  - Inclusion use cases may be complete use cases, or they may just specify a fragment of behaviour for inclusion elsewhere



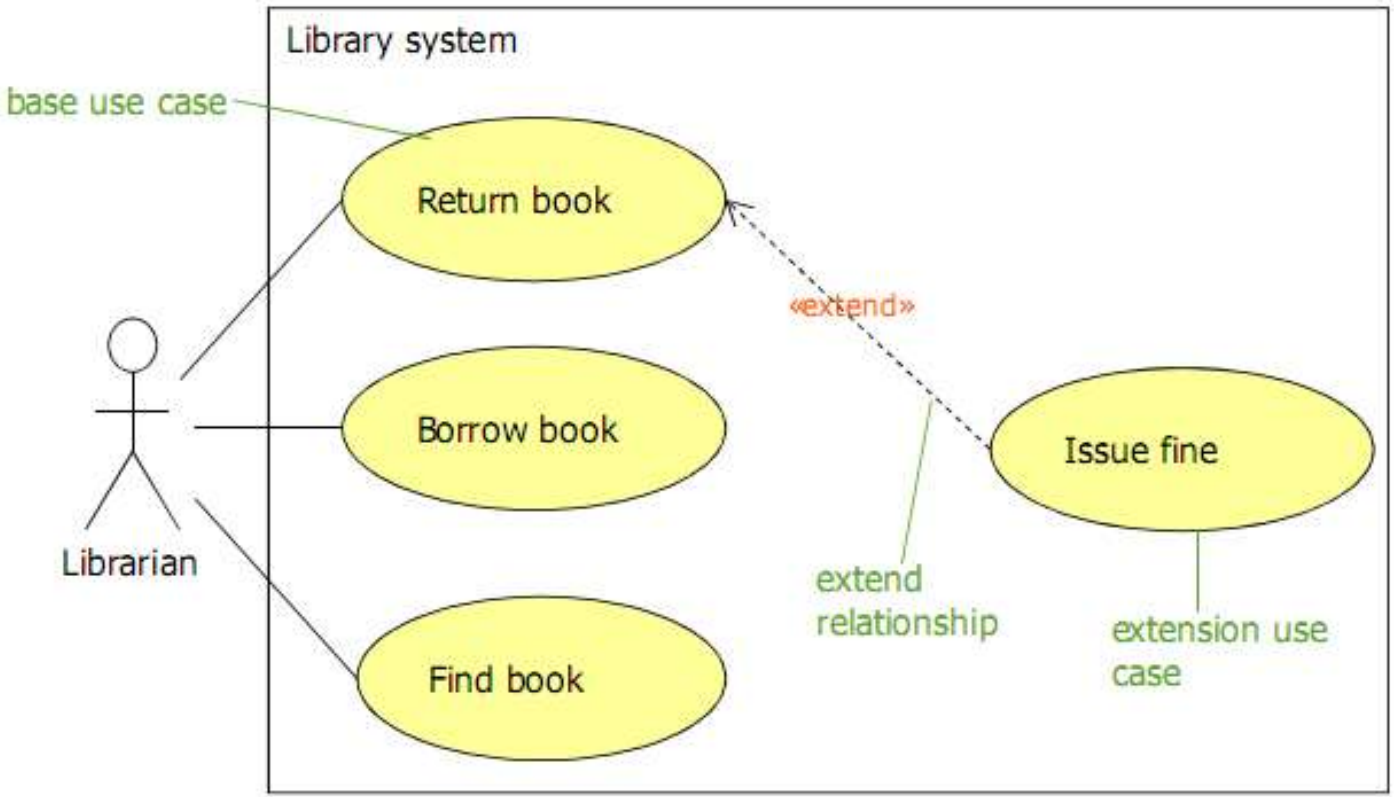
Use Case -- <<include>> example



Use Case Extension

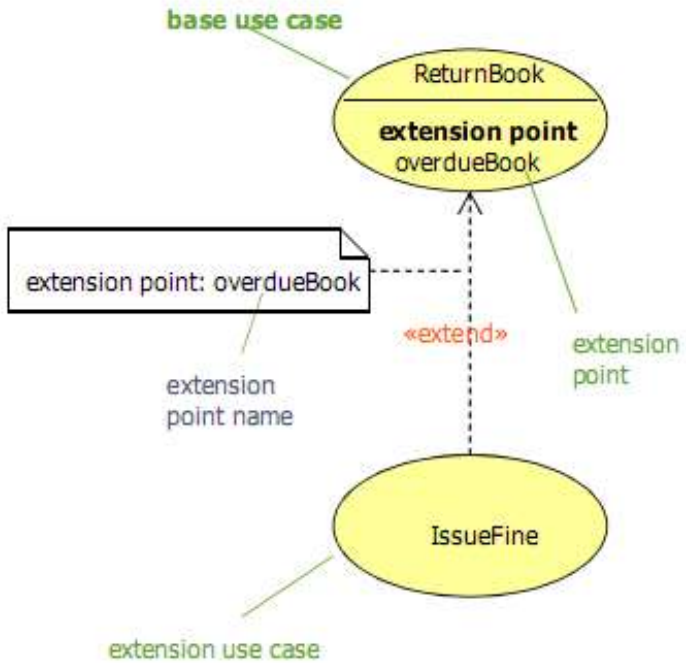
- «extend» is a way of adding new behaviour into the base use case by inserting behaviour from one or more extension use cases
  - The base use case specifies one or more extension points in its flow of events

Use Case Extension

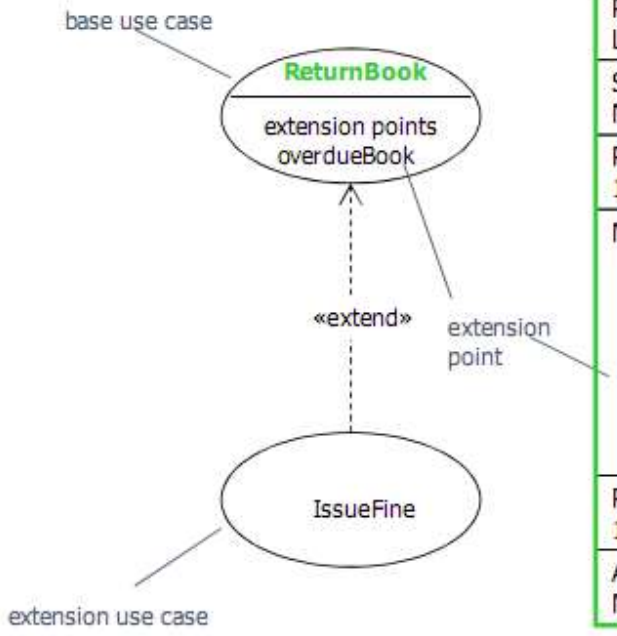


Use Case Extension

- The «extend» relationship may specify which of the base use case extension points it is extending



Use Case – Base use case



Use case: ReturnBook
ID: 9
Brief description: The Librarian returns a borrowed book.
Primary actors: Librarian
Secondary actors: None.
Preconditions: 1. The Librarian is logged on to the system.
Main flow: 1. The Librarian enters the borrower's ID number. 2. The system displays the borrower's details including the list of borrowed books. 3. The Librarian finds the book to be returned in the list of books. **extension point: overdueBook,** Extension: Issue Fine 4. The Librarian returns the book. ...
Postconditions: 1. The book has been returned.
Alternative flows: None.

- Extension points are ***not*** numbered, as they are ***not*** part of the flow

Use Case – Extension use case

Use case: ReturnBook
ID: 9
Brief description: The Librarian returns a borrowed book.
Primary actors: Librarian
Secondary actors: None.
Preconditions: 1. The Librarian is logged on to the system.
Main flow: 1. The Librarian enters the borrower's ID number. 2. The system displays the borrower's details including the list of borrowed books. 3. The Librarian finds the book to be returned in the list of books. **extension point: overdueBook,** Extension: Issue Fine 4. The Librarian returns the book. ...
Postconditions: 1. The book has been returned.
Alternative flows: None.

Extension Use case: IssueFine
ID: 10
Brief description: Segment 1: The Librarian records and prints out a fine.
Primary actors: Librarian
Secondary actors: None.
Segment 1 preconditions: 1. The returned book is overdue.
Segment 1 flow: 1. The Librarian enters details of the fine into the system. 2. The system prints out the fine.
Segment 1 postconditions: 1. The fine has been recorded in the system. 2. The system has printed out the fine.

- Extension use cases have one or more *insertion segments* which are behaviour fragments that will be inserted at the specified extension points in the base use case

Exercise – ABC Rental

The scope of the project for ABC Rental Processing system is to provide rental/return processing for videos, including customer maintenance, video inventory maintenance, and historical information maintenance. At the end of the day, accounting totals of sales information are generated, but there is no automated accounting interface. There is no purchase order processing in this application. The application's main function is rental processing.

When a customer selects a video, he/she first tells the clerk their phone number. The clerk uses the phone number to 'look up' the customer and validate the rentals. If the customer is new (not on file), the customer information is entered and stored. After phone number processing, the customer either gives the clerk the cardboard shell, or tells the clerk the video name.

After entering the information into the computer system, the clerk needs to provide some record with customer signature that the rental took place. This record accounts for the transaction and establishes customer liability for the rental property.

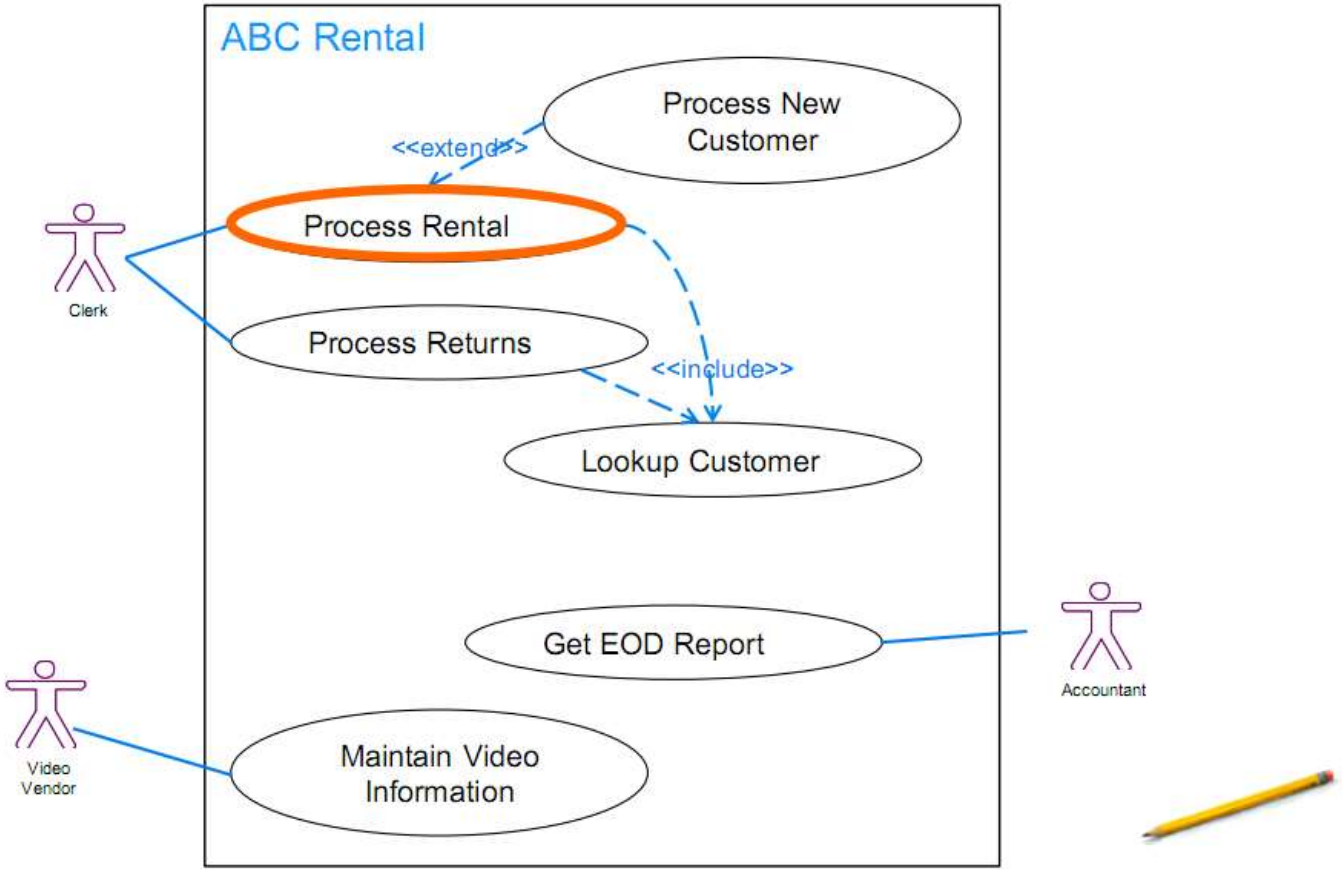
When the tape is returned, the charges are computed based on the due date of the rental(s).

The information about a new video is entered into the system by the video vendor.

The accountant does not feed any information into the application, and receives only an end-of-day rental summary.

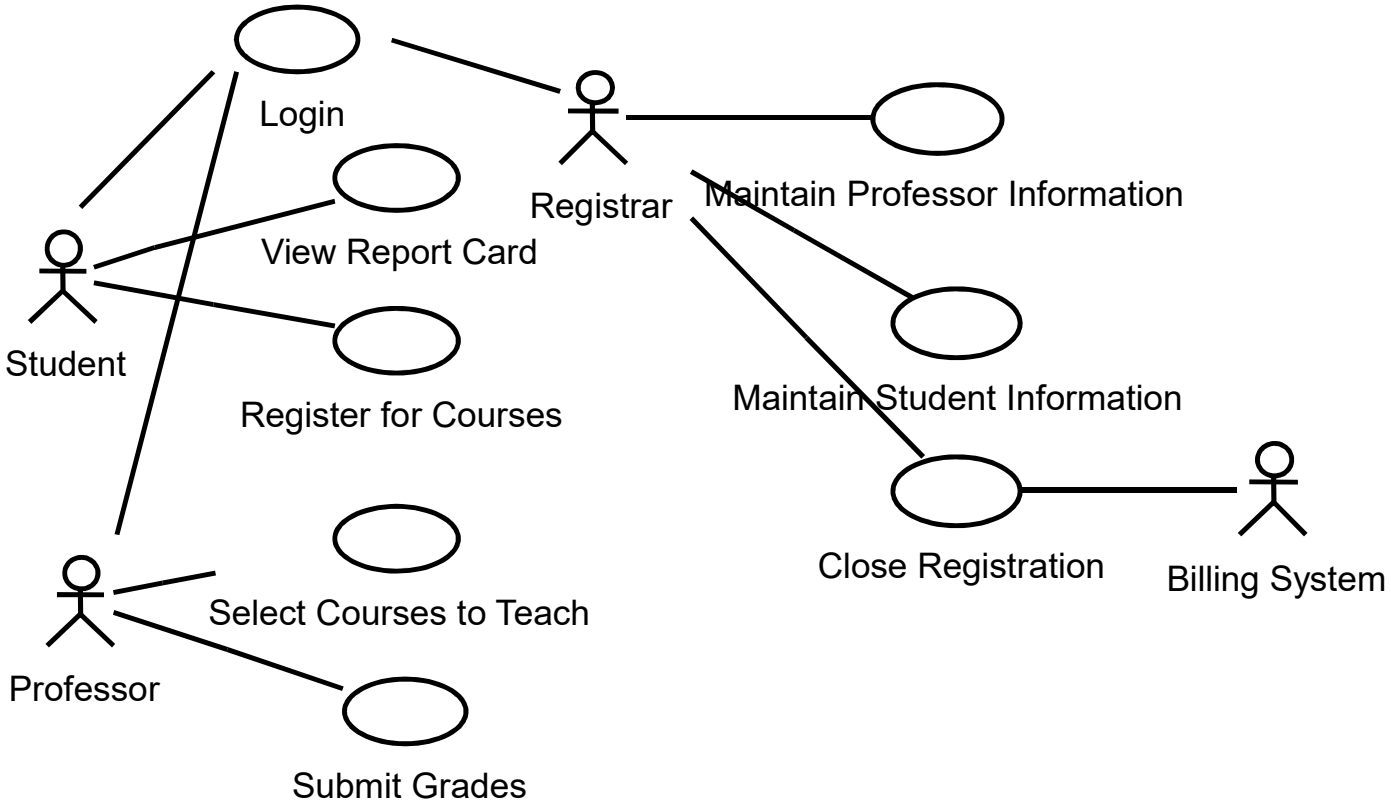
Draw a Use Case Diagram for the **ABC Rental Processing**

ABC Use Case Diagram

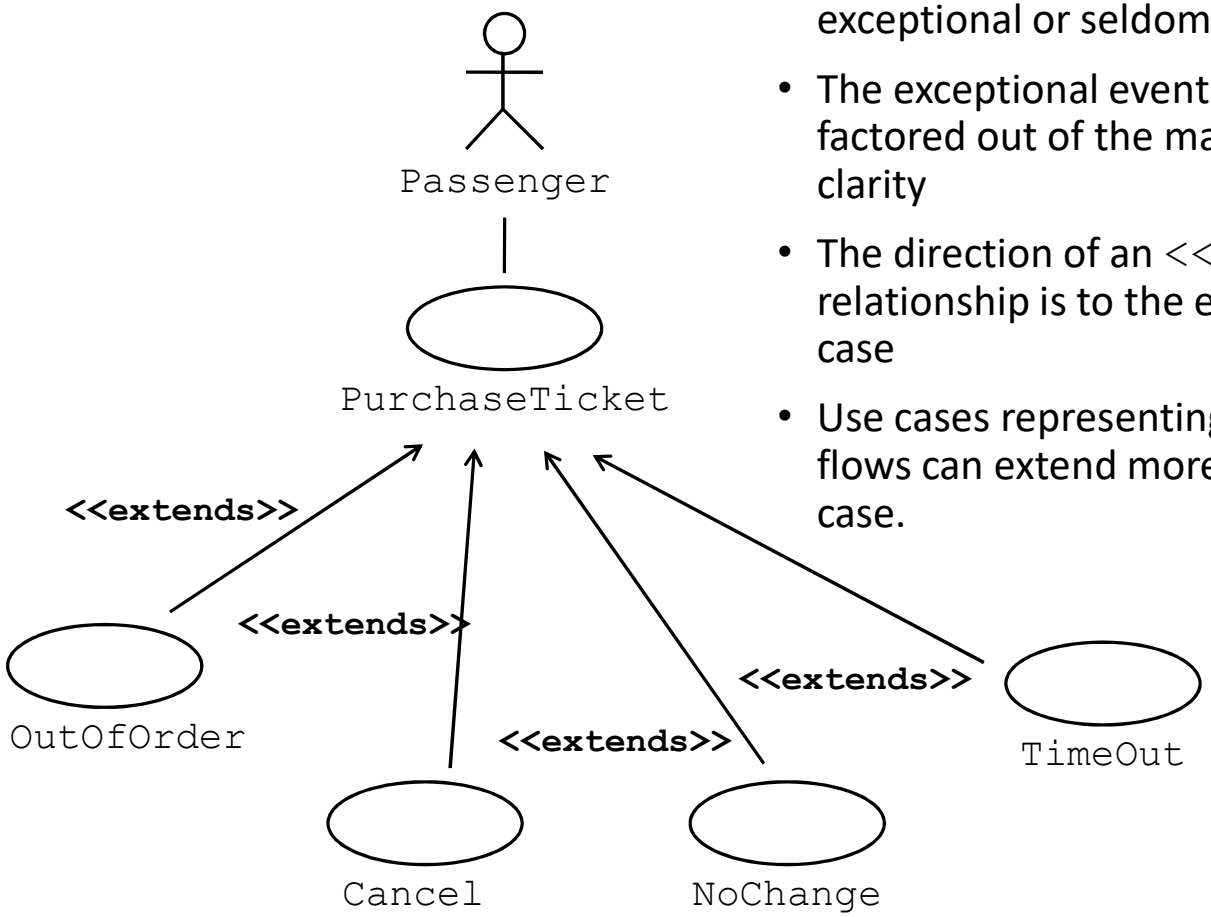


Use Cases – Examples

A Sample Use Case Diagram: A University Course Registration System

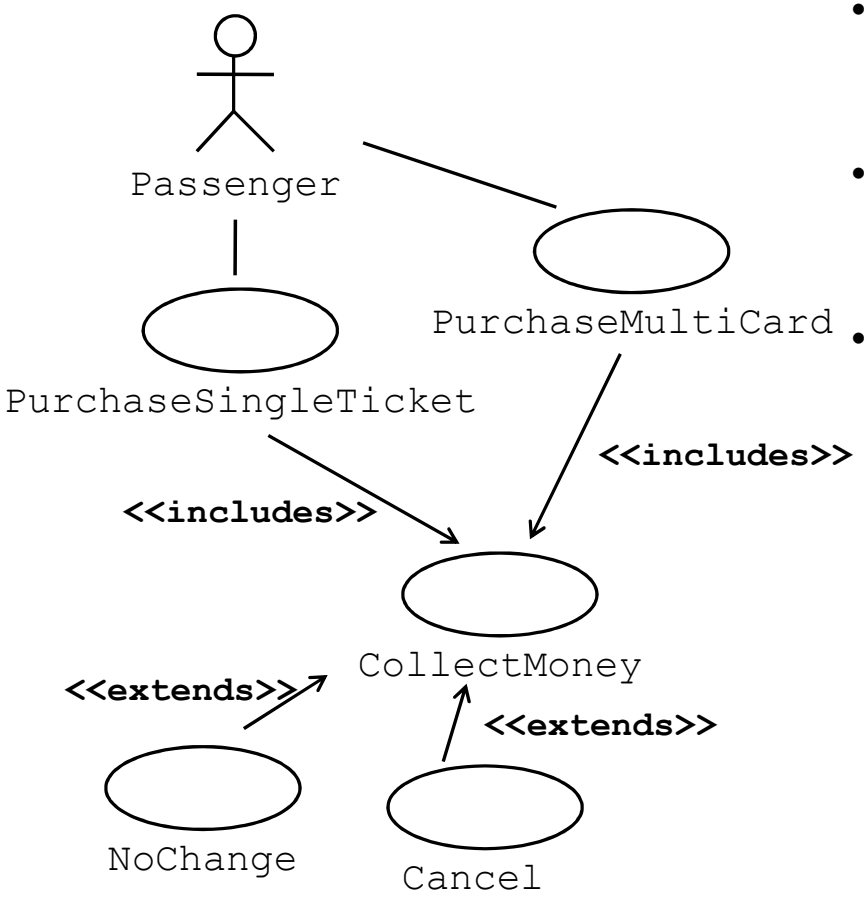


The *<<extends>>* Relationship



- `<<extends>>` relationships model exceptional or seldom invoked cases
- The exceptional event flows are factored out of the main event flow for clarity
- The direction of an `<<extends>>` relationship is to the extended use case
- Use cases representing exceptional flows can extend more than one use case.

The *<<includes>>* Relationship



- <<includes>> relationship represents common functionality needed in more than one use case
- <<includes>> behavior is factored out for reuse, not because it is an exception
- The direction of a <<includes>> relationship is to the using use case (unlike the direction of the <<extends>> relationship).

Scenarios

- A sequence of interactions between the user and the system.
- To achieve a specified goal
- Each use case represents a group of scenarios
- Each scenario describes a different sequence of events involved in achieving the goal

Successful scenario – Wheels

- *Stephanie chooses a mountain bike*
- *Annie sees that its number is 468*
- *Annie enters this number into the system*
- *The system confirms that this is a woman's mountain bike and displays the daily rate (£2) and the deposit (£60)*
- *Stephanie wants to hire the bike for a week*
- *Annie enters this and the system displays the cost*
- *Stephanie agrees this*
- *Annie enters Stephanie's details*
- *Stephanie pays the £74*
- *Annie records this and the system prints out a receipt*

Scenarios

- A successful scenario, one that achieves the use case goal, is sometimes referred to as
- a ‘happy day’ scenario or
- the ‘primary path’.

Scenarios

Scenario for the situation where the use case goal is not achieved

- *Michael arrives at the shop at 12.00 on Friday*
- *He selects a man's racer*
- *Annie see the number is 658*
- *She enters this number into the system*
- *The system confirms that it is a man's racer and displays the daily rate (£2) and the deposit (£55)*
- *Michael says this is too much and leaves the shop without hiring the bike.*

The scenarios should document:

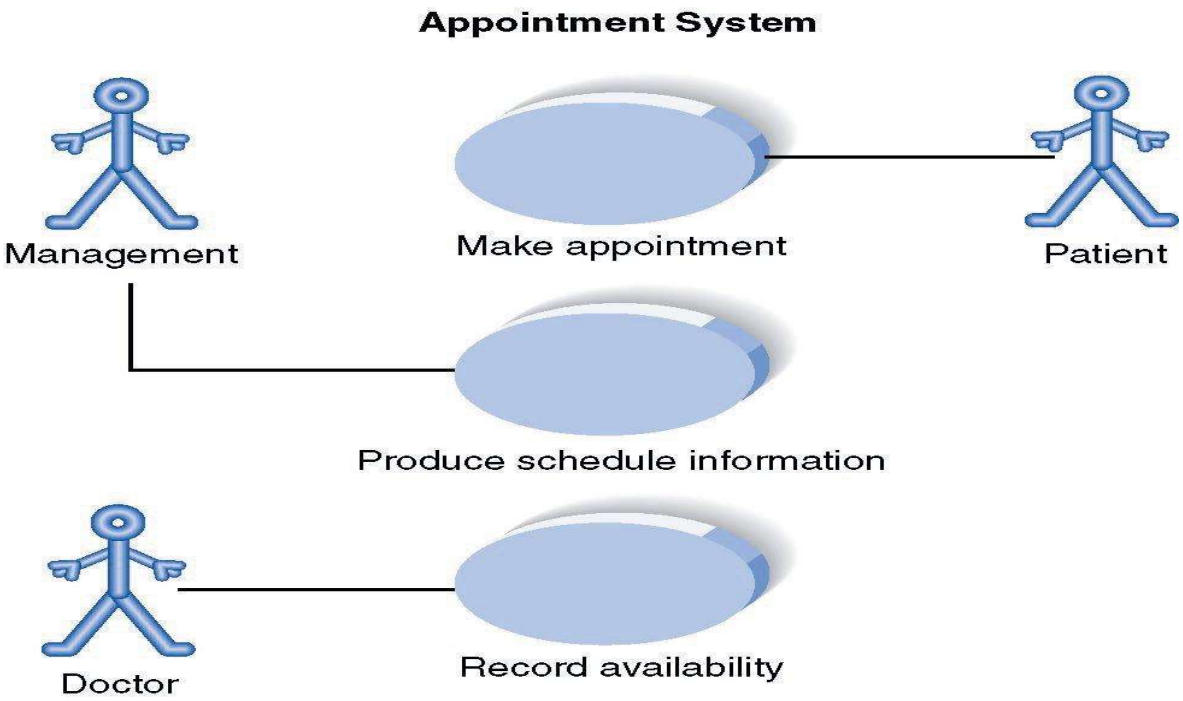
- a typical sequence of events leading to the achievement of the use case goal – e.g. a customer hires a bike
- obvious variations on the norm, e.g. a customer hires several bikes
- sequences of events where the use case goal is not achieved e.g. the customer cannot find the bike he wants

Use case : Issue bike
Actors: Receptionist
Goal: To hire out a bike
Overview:
When a customer comes into the shop they choose a bike to hire. The receptionist looks up the bike on the system and tells the customer how much it will cost to hire the bike for a specified period. The customer pays, is issued with a receipt, then leaves with the bike.
Cross reference R3, R4, R5, R6,R7, R8, R9, R10
Typical course of events

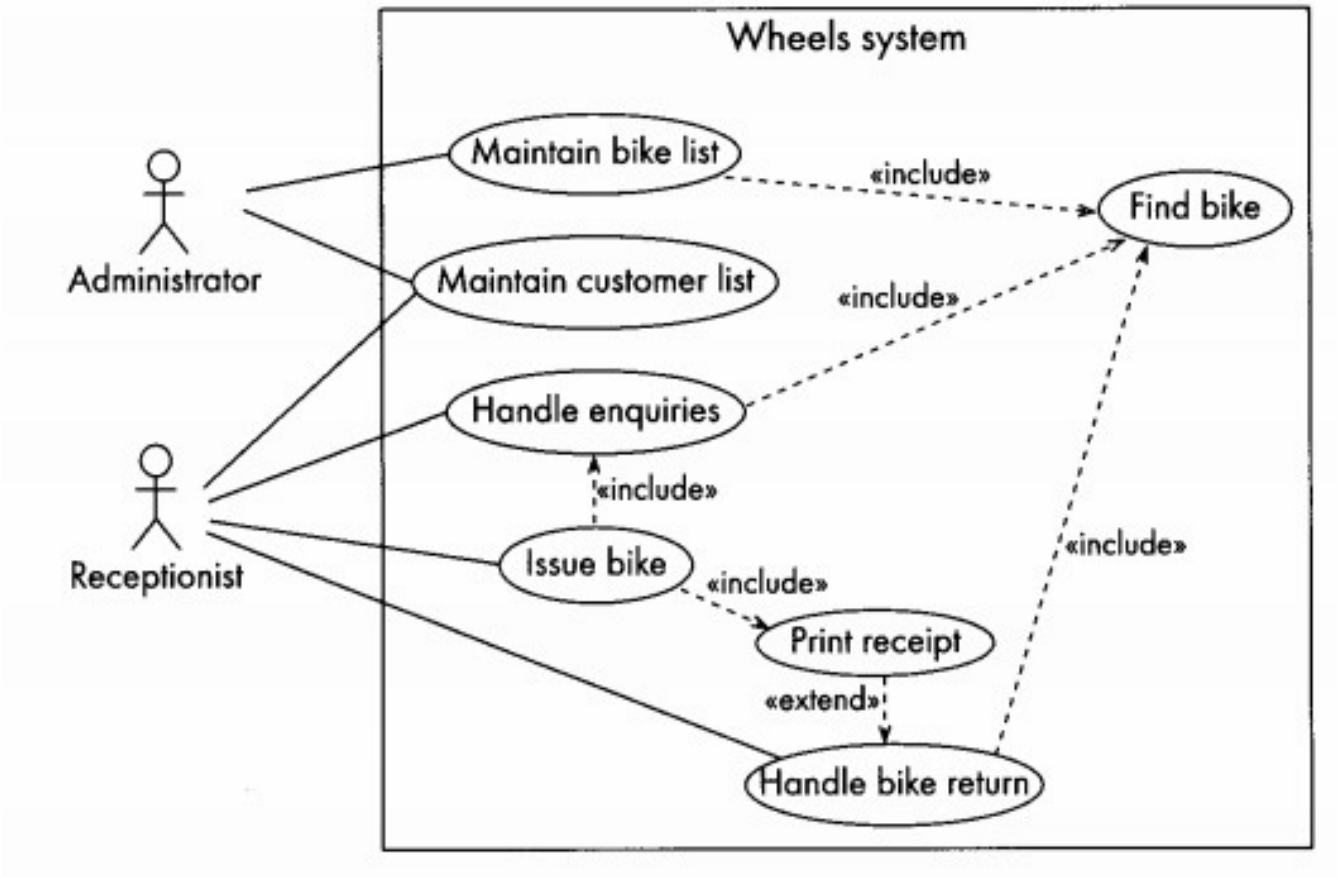
Actor action	**System response**
1. The customer chooses a bike	
2. The Receptionist keys in the bike number	3. Displays the bike details 4.
Customer specifies length of hire	
5. Receptionist keys this in	6. Displays total hire cost
7. Customer agrees the price	
8. Receptionist keys in the customer details	9. Displays customer details
10. Customer pays the total cost	
11. Receptionist records amount paid	12. Prints a receipt

Alternative courses
Steps 8 and 9. The customer details are already in the system so the Receptionist needs only to key in an identifier and the system will display the customer details.
Steps 7 – 12. The customer may not be happy with the price and may terminate the transaction.

Use Case Diagram for Appointment System (Level of Information)



Wheels System with includes and extends



Scenario example : Warehouse on Fire

- Bob, driving down main street in his patrol car notices smoke coming out of a warehouse. His partner, Alice, reports the emergency from her car.
- Alice enters the address of the building into her wearable computer , a brief description of its location (i.e., north west corner), and an emergency level.
- She confirms her input and waits for an acknowledgment.
- John, the dispatcher, is alerted to the emergency by a beep of his workstation. He reviews the information submitted by Alice and acknowledges the report. He allocates a fire unit and sends the estimated arrival time (ETA) to Alice.
- Alice received the acknowledgment and the ETA.

Observations about Warehouse on Fire Scenario

- Concrete scenario
  - Describes a single instance of reporting a fire incident.
  - Does not describe all possible situations in which a fire can be reported.

- Participating actors
  - Bob, Alice and John

Other Scenarios Possibilities for an Incident Management System

- What needs to be done to report a “Cat in a Tree” incident?
- Who is involved in reporting the incident?
- What does the system do, if no police cars are available? If the police car has an accident on the way to the “Cat in a Tree” incident?
- What do you need to do if the “Cat in the Tree” turns into a “Grandma Has Fallen From the Ladder”?
- Can the system cope with simultaneous incident reports “Cat in the Tree” and “Warehouse on Fire?”

Use Case Model for Incident Management