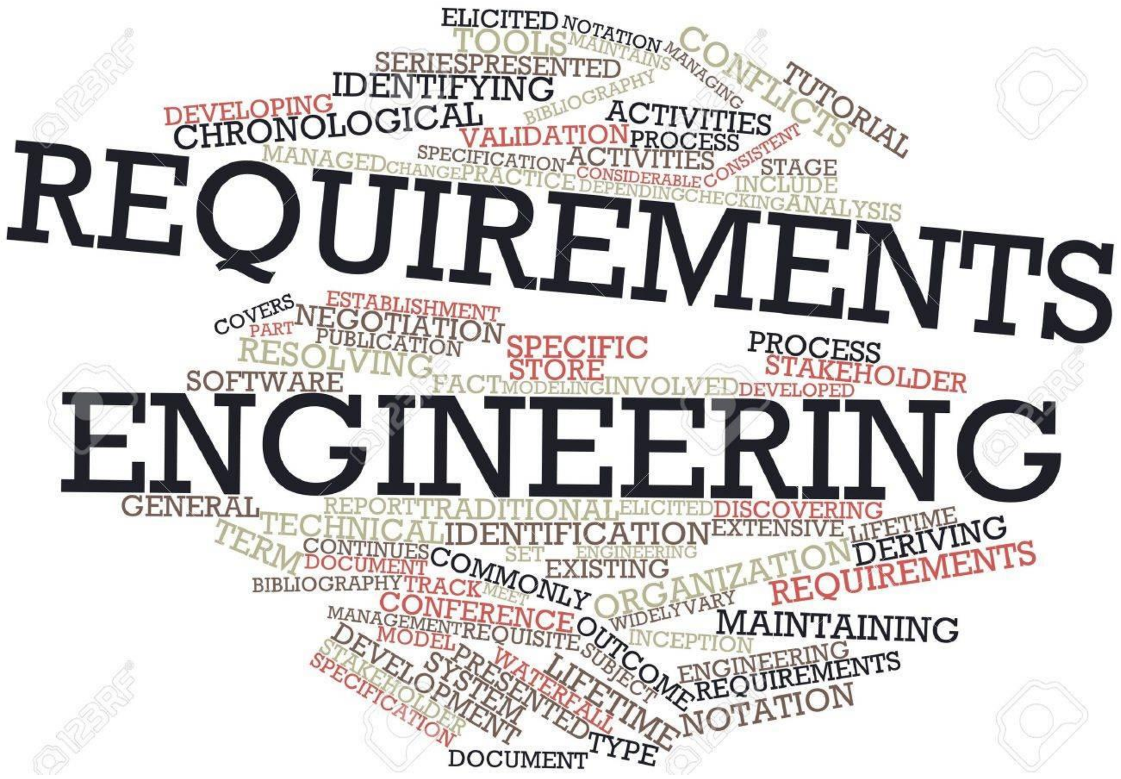


SWE 215: Software Requirements Engineering

Lecture 6

Fundamentals of Scenarios

Course Topics

- ~~Why Requirements Engineering?~~
- ~~Introduction to Requirements~~
- ~~RE in Software Development Life Cycles~~
- ~~System Vision, Context, and RE Framework~~
- ~~Fundamentals of Goal Orientation~~
- Fundamentals of Scenarios
- Requirements Discovery
- User Stories and Agile Estimation
- Features Prioritization
- Requirements Negotiation
- Requirements Validation
- Fundamentals of Requirements Management

Lecture Objectives

- Fundamentals of scenarios in Requirements Engineering
- Main scenario types and their use in Requirements Engineering
- UML Use Case Diagrams
- UML Activity Diagrams

What is a Scenario ?

- **Goals** alone **do not sufficiently support requirements elicitation**
- Stakeholders typically find it easier to communicate their requirements in terms of examples (e.g., **interaction sequences**) rather than **abstract intentions**
- A **scenario** describes a **concrete example** of **satisfying** or **failing** to satisfy a **goal** (or set of goals).
- A scenario typically defines **a sequence of interaction steps** executed to satisfy the goal and relates these interaction steps to the system context.

Scenarios as a Means for Putting Requirements in Context

Scenarios are well suited for documenting **context information**.

Kinds of context information:

- **Actors**: Persons or systems interacting with the system
- **Roles**: Specific class of actors
- **Goals**: Scenarios illustrate satisfaction of goals
- **Precondition**: define conditions that must hold before executing the scenario
- **Post conditions**: must hold within the system or context after executing the scenario
- **Resources**: special preconditions referring to persons, information, financial or other material resources needed for a scenario

Positive and Negative Scenarios

- Scenario can be **positive** or **negative** (regarding the satisfaction of a goal)

- Positive scenario documents sequence of interactions leading to **the satisfaction of a goal**

- Negative scenario documents sequence of interactions **failing to satisfy a goal**
  - Negative scenario may be **allowed** or **forbidden**

- **Positive and negative scenarios complement each other**

Allowed Negative Scenarios

<u>**Example**</u>: a scenario **in which an actor provides an incorrect input to the system**. Due to this incorrect input, **the system cannot satisfy some goal**, but still has to react accordingly to the incorrect input.

Example: "Enable customers to withdraw cash from their account." Chris inserts his bank card into the slot of the ATM (automated teller machine). Chris enters his personal identification number and the amount to withdraw. The ATM informs Chris that withdrawing the desired amount is not possible because the amount exceeds his balance.

In the example, the **stated goal is not satisfied** due to **an insufficient account balance**.

Forbidden Negative Scenarios (failure scenario)

- The failure to satisfy the related goals **cannot be tolerated**
- For instance, because the respective **goals are critical** for the system and/or its context, the execution of the sequence is regarded as a **system failure**.
- The stakeholders must take appropriate measures to **prevent the execution of the sequence of interactions documented in a forbidden negative scenario.**

Forbidden Negative Scenarios

- **Example**: "Enable customers to withdraw cash from their account".

Jack inserts his bank card into the slot of the ATM. Jack enters his personal identification number and the amount to withdraw. The ATM **charges the desired amount from Jack's account**. When dispensing the money, the **Dispensing mechanism of the ATM fails**.

In the scenario, the customer's account is charged although the ATM Dispenses no money. Clearly this behavior of the ATM is not tolerable.

Descriptive and Exploratory/Explanatory Scenarios

- **Descriptive scenario**:
  - Describes **process** or **workflow**
  - **Purpose**: understanding its operations

- **Exploratory/Explanatory scenario**:
  - Explore and evaluate possible, alternative solutions in order to support the selection of **one alternative solution**.
  - **Provides background information and rationales for particular interaction sequences**
  - For example, 'automatic braking manoeuvre' …… there is a high risk of rear-end collision. A rapid change of lane might cause the car to skid or spin……………

Instance and Type Scenarios

Instance scenario: Describes a **concrete** (existing or envisioned) **sequence of interactions between concrete actors**

Carl wants to **drive** to **Union Street in Plymouth**. Carl uses the navigation system of his **VW Golf with license number 'E-12'**. Carl selects **'enter destination' in the main menu**, enters the destination **'Union Street, Plymouth'**, and presses the **key 'calculate route**.

Type scenario: **Abstracts** from the concrete actors, inputs and outputs of a specific sequence of interactions

The **driver** wants to drive to a **destination** using the navigation system. He enters the **destination**. The system **calculates the route** from the current position of the car to the entered destination.

Instance and Type Scenarios

- **Mixed scenario**
  - Important content described at **instance level**
  - **Content not completely understood** is described at **instance level** to **avoid errors resulting from early abstractions**
  - Content that is **well understood** described at the **type level**
  - **Conflicting** or **potentially conflicting content** described at **instance level**

The driver activates the navigation system. The system asks the driver: ‘Please state your destination or press the ‘enter destination’ button on the main menu. The driver presses the ‘enter destination’ button and enters the destination manually. After entering the destination the driver initiates the calculation of the route.

System-Internal, Interaction, and Context Scenarios

- **System-internal (type A) scenarios**

Focus **exclusively** on **system-internal interactions** (interactions that occur within the system boundaries)

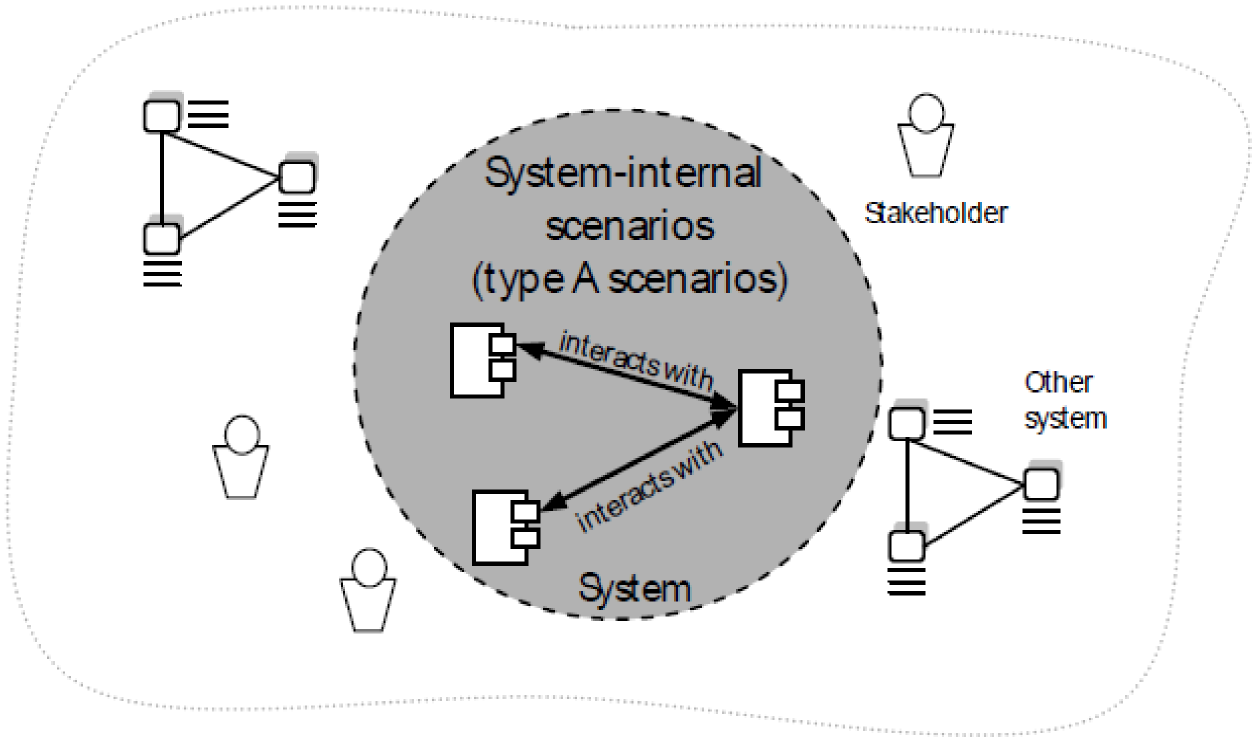
- **Interaction (type B) scenarios**

Document sequences of **interactions between system and its actors** (persons and systems in the context)

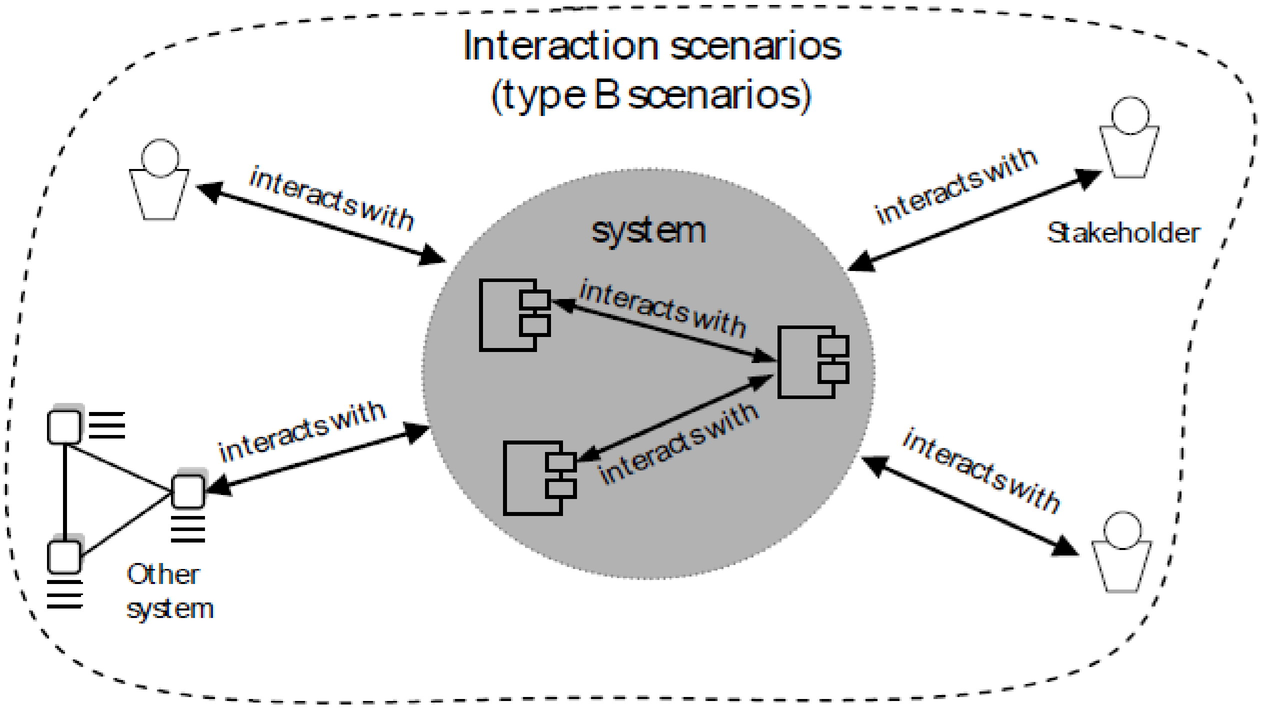
- **Context (type C) scenarios**

Document additional context information relevant for the system usage

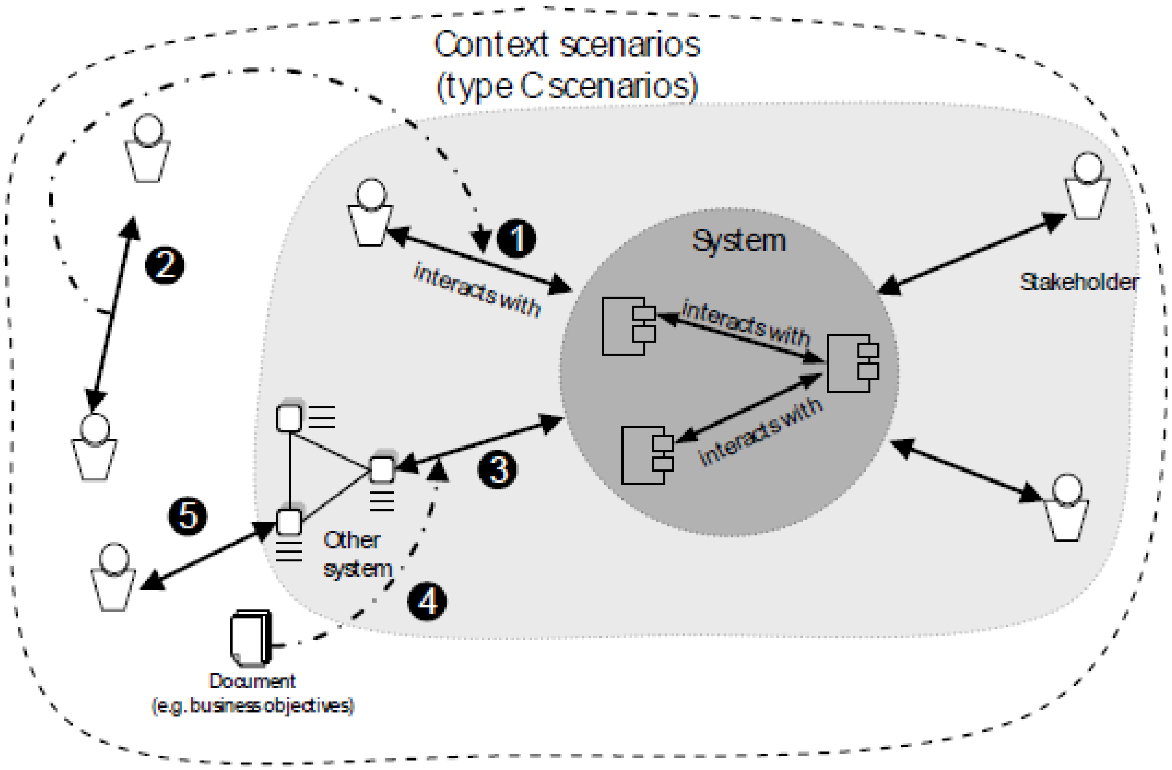
System-internal scenarios (type A scenarios)



Interaction scenarios (type B scenarios)



Context scenarios (type C scenarios)



Primary versus Secondary Actors

Primary Actors: The Actor(s) using the system to **achieve a goal**. The Use Case **documents the interactions between the system and the actors to achieve the goal of the primary actor.**

Secondary Actors: Actors that the system **needs assistance from** to achieve the primary actor's goal.

[Cockburn 2006]

Secondary actors **may or may not have goals** that they expect to be satisfied by the use case, the primary actor **always has a goal**, and the use case exists to satisfy the primary actor.

Example I

A bank loan officer wants to review a loan application from a customer, and part of the process involves a real-time credit rating check.

- **Use Case Name**: Review Loan Application
- **Primary Actor**: Loan Officer
- **Secondary Actors**: Credit Rating System

We need to define the Secondary Actor because without the "Credit Rating System" we cannot successfully complete the Use Case.

In other words, the goal of the Primary Actor is to successfully complete the Loan Application, but they need the explicit "help" of the Secondary Actor (Credit Rating System) to achieve this goal.

Example II

A Human Resources manager wants to change the job code of an employee, and as part of the process, automatically notify several other departments within the company of the change.

- **Use Case Name:** Maintain Job Code
- **Primary Actor:** Human Resources Manager
- **Secondary Actors:** None

This is where people sometimes get confused.

Example II (cont.)

We don’t include The “other departments” as **Secondary Actors** for the following reasons:

1. The other departments are not required for the successful completion of the Use Case
2. We are not expecting any response from the other departments (at least within the bounds of the Use Case under discussion)

Within the detail of the Use Case Specification Main Success Scenario, we would include something like: “The system sends a notification to the related department heads (ref. Business Rule BR101)”

UML Use Case Diagram (UCD)

Use Case Diagram (UCD)

- Built in early stages of development

- **Purpose**:
  - Model the **context of a system**
  - Capture the requirements of a system

- Diagram that shows **a set of use cases and actors and their relationships.**
- Actors may be connected to use cases only by **association**.
- An association between an actor and a use case indicates that the actor and the use case **communicate with one another**, each one possibly sending and receiving messages.

- However, they do not capture the full information of the actual use cases → **textual description is essential**

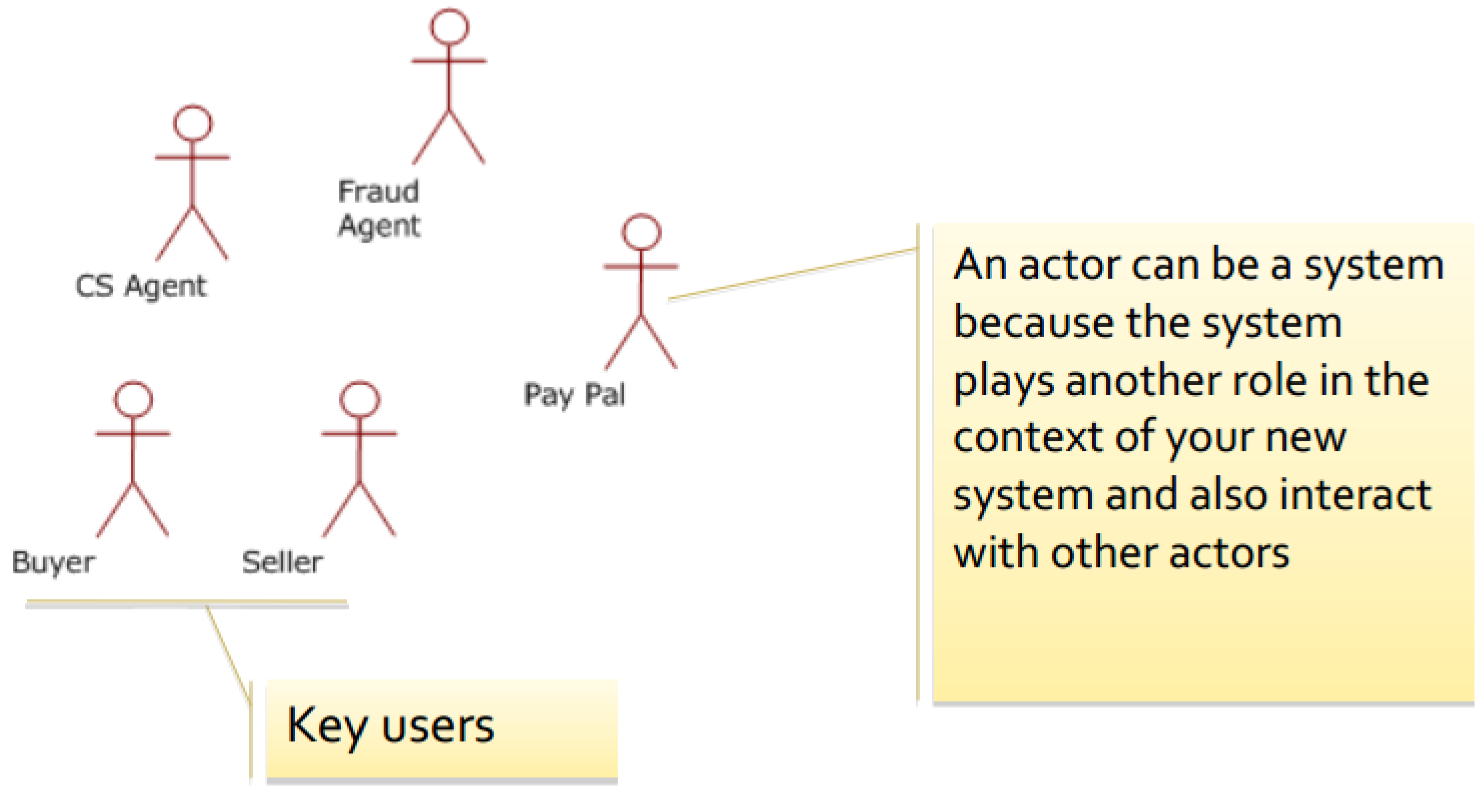
UML: What is an Actor?

The Unified Modeling Language (UML) defines an Actor as:
"An actor specifies a role played by a user or any other system that interacts with the subject"

- Actors **are NOT part of the system**
- Include system components only if they are responsible for initiating/triggering a use case.
- For example, a **timer that triggers sending of an e-mail reminder**
- Include all **user roles** that interact with the system
- Actors are **not individual persons** (e.g., John) but stimulates the system to react (primary actor)

Actors in UML

In UML, an actor is represented as a stickman.


Fraud Agent
CS Agent
Pay Pal
Buyer
Seller
Key users
An actor can be a system because the system plays another role in the context of your new system and also interact with other actors

What is a use case?

- A **case of a use** of the system/product
- **Tells a story:** A use case is a **description** of **a sequence of events** involving **interactions of a user with the system**

- A use case describes **what a system does** but it does not specify how it does it.

- A use case typically represents a major piece of **functionality** that is **complete** from beginning to end.

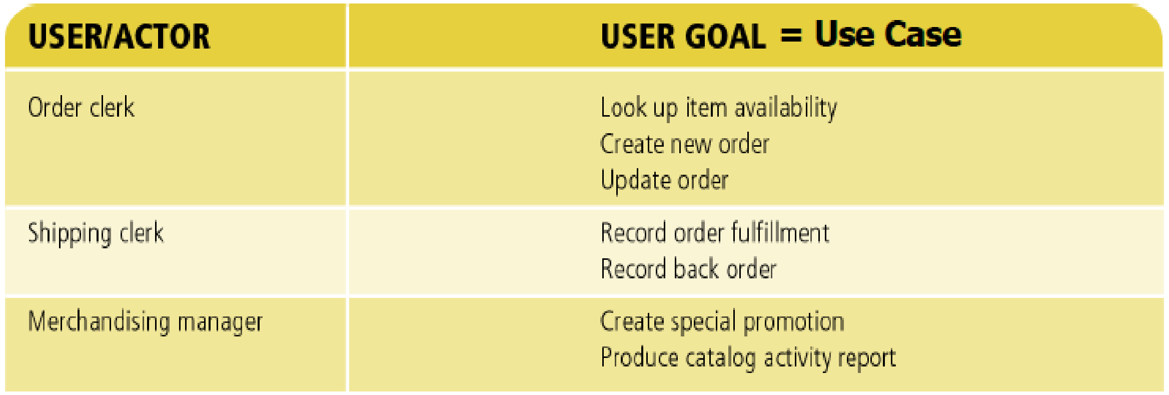
- **Is oriented toward satisfying a user's goal**

Basic elements of use case diagram

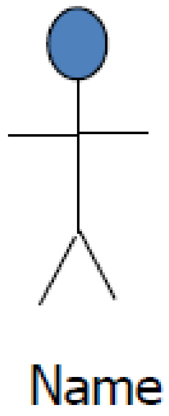
Actor: is someone interacting with a use case (system function). Named by a **noun.**

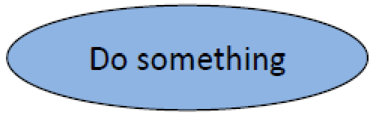
Use Case: specifies system function (process – automated or manual). Named by a **verb** (Do something)

Each Actor **must be linked to a use case, while some use cases may not be linked to actors.**



USER/ACTOR	USER GOAL = Use Case
Order clerk	Look up item availability Create new order Update order
Shipping clerk	Record order fulfillment Record back order
Merchandising manager	Create special promotion Produce catalog activity report

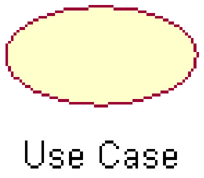




How do we describe use cases?

- Textual or tabular descriptions
- Use Case Diagrams

In the UML, a use case is represented as an oval.



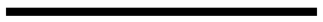
Use Case Template

Use Case ID:			
Use Case Name:	Each use case is given a name.		
Created By:	Author	**Last Updated By:**	
Date Created:		**Last Revision Date:**	
Actors:	Actors associated with this use case		
Description:	A brief description of the use case, typically one or two sentences.		
Dependencies	Description of whether the use case depends on other use cases, that is, whether it includes or extends another use case.		
Trigger:	Describes how the use case is triggered.		
Preconditions:	One or more conditions that must be true at the start of the use case		
Postconditions:	Condition that is always true at the end of the use case if the main sequence has been followed.		
Normal Flow:	Description of the main sequence of the use case		
Alternative Flows:	Description of alternative branches off the main sequence.		
Frequency of Use:	How frequent this use case is used		
Special Requirements:	Any special requirements		
Assumptions:	Any assumptions		
Notes and Issues:	Any extra notes and issues		

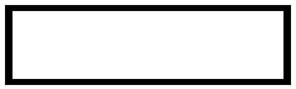
Identifying Use Cases

- **What will the actor use the system for?**
  - Describe the functions that the user will want from the system
- **Will the actor create, store, change, remove, or read data in the system?**
  - Describe the operations that create, read, update, and delete information
- **Will the actor need to inform the system about external events and vice versa?**
  - Describe how actors communicate information about events that the system must know about
  - Describe how actors are notified of changes to the internal state of the system

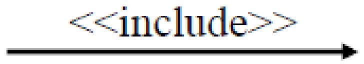
Elements of a Use Case Diagram



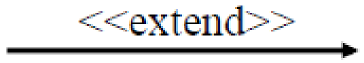
Connection between Actor and Use Case



Boundary of system



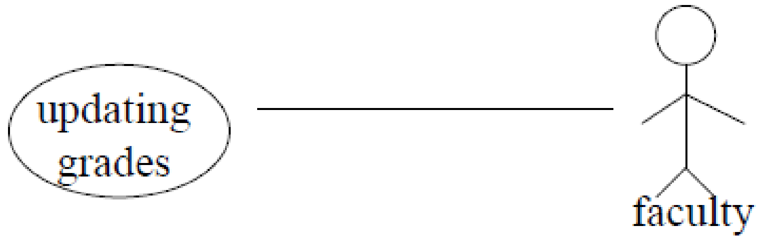
Include relationship between Use Cases (one UC must call another; e.g., Login UC includes User Authentication UC)



Extend relationship between Use Cases (one UC calls Another under certain condition; think of if-then decision points)

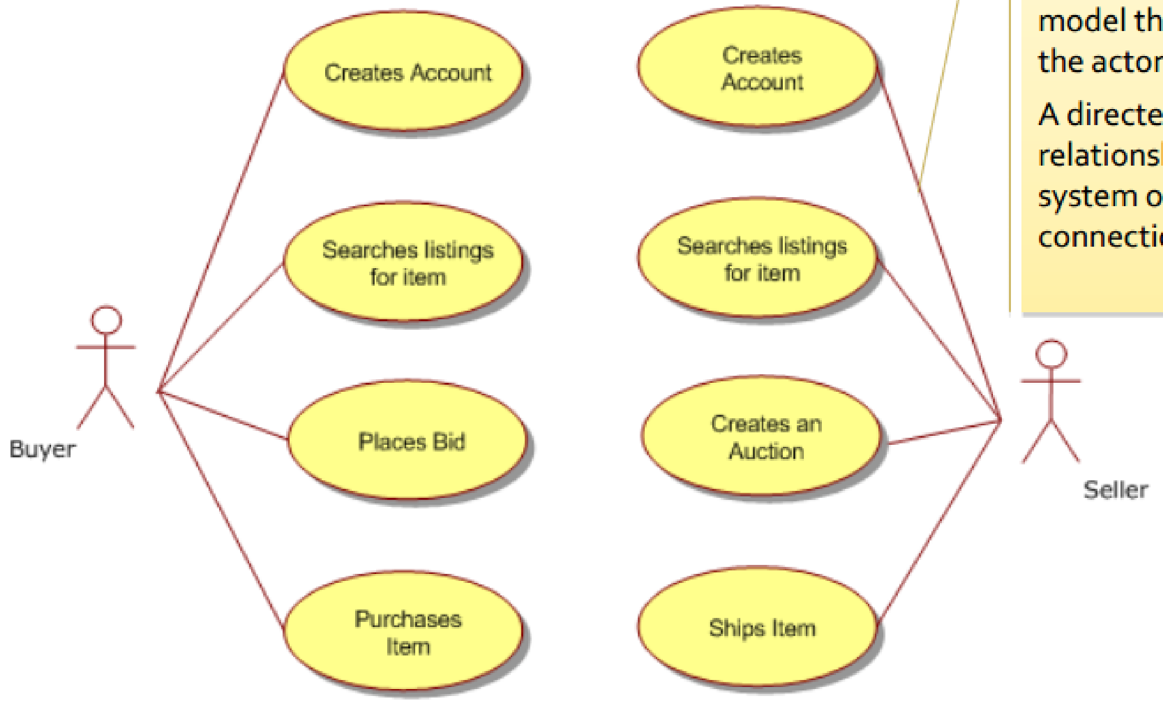
Relationship between Use Cases and Actors

Actors may be connected to use cases by **associations**, indicating that the actor and the use case communicate with one another.



Relationship between Use Cases and Actors

Define your use case Actor Goals

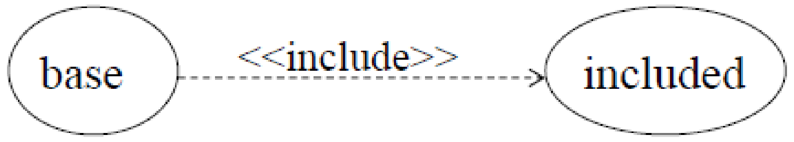


Note:
Association relationships only show which actors interact with the system to perform a given use case.

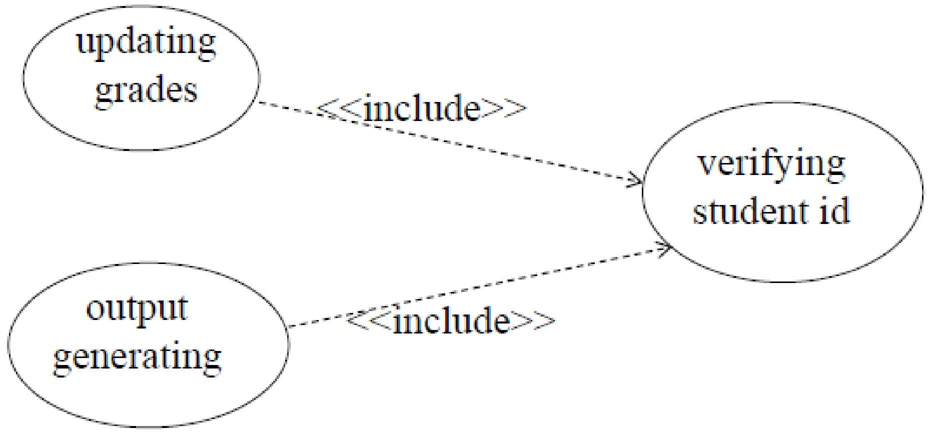
Association relationship **DO NOT** model the flow of data between the actor and the system.

A directed association relationship only shows if the system or the actor initiates the connection.

Include Relationship



- The base use case explicitly incorporates the behavior of another use case at a location specified in the base.
- **The included use case never stands alone**. It only occurs as a part of some larger base that includes it.
- **Reuse:** Enables us to **avoid describing the same flow of events several times by putting the common behavior in a use case of its own**.

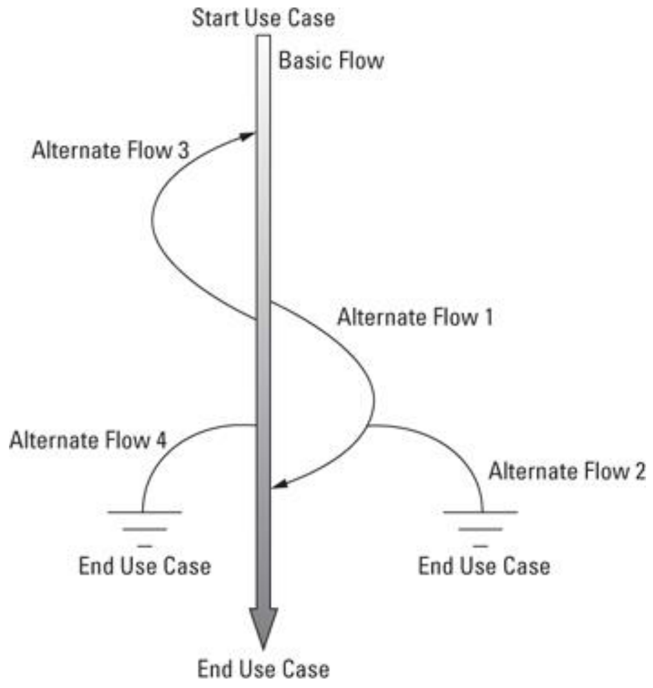


Include Relationship

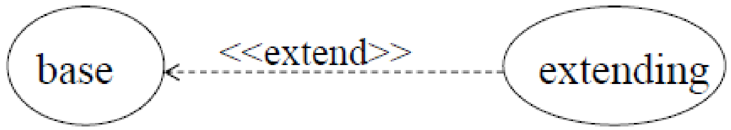
- A standard case linked to a ***mandatory*** use case.
  - **Example**: to Authorize Car Loan (standard use case), a clerk must run Check Client's Credit History (include use case).
- Standard use case can NOT execute without the include case → **tight coupling**.
- The standard UC **includes** the mandatory UC (use the verb to figure direction arrow).

Basic and Alternative Flows

- **Basic flow**: Flow that represents the most common path (the "happy path") from start to finish through the use case.
- What actor's event starts the use case?
- How does the use case end?
- How does the use case repeat some behavior?
- **A number of alternate flows based on both regular circumstances and exceptional events**. The following questions can help discover these paths.
- What else can the actor do?
- How will the actor react to optional situations?
- What variants might happen?
- What exceptions to the usual behavior may occur?



Extend Relationship

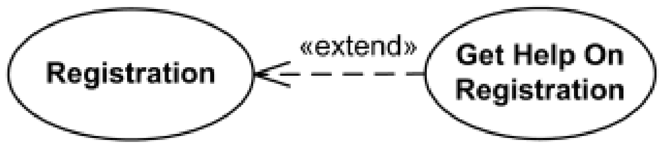


- **Extended** use case is meaningful on its own, it is **independent** of the extending use case.
- **Extending** use case typically defines **optional behavior** that is not necessarily meaningful by itself.
- The extension takes place at one or more **extension points** defined in the **extended use case**.
- **Extend** relationship is shown as a dashed line with an open arrowhead directed from the **extending use case** to the **extended (base) use case**. The arrow is labeled with the keyword **«extend»**.

Extend Relationship

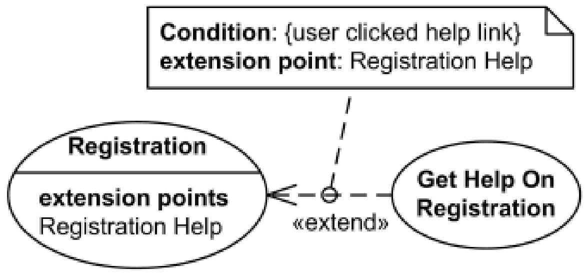
- The new functionality may open up a whole raft of possibilities and **there is a danger that the Alternative Flow spawns further sub flows**.
- The Use Case may become difficult to manage. To avoid this the **«extend» relationship** can be used to pull **the Alternative Flow and its sub-flows out into a new Use Case.**
- The «extend» relationship says that we execute the base Use Case but when we get to a specified point in the flow, if the right conditions are met, we perform some different steps.
- Clearly this is very similar to an Alternative Flow. The advantage is that the Alternative Flow and any dependent sub-flows have been moved into a separate Use Case.

Extend Relationship: Example



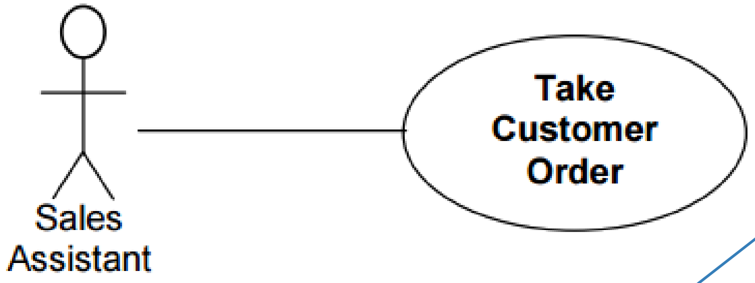
Registration *use case is complete and meaningful on its own.*
It could be extended with optional ***Get Help On Registration*** *use case.*

The **condition** of the extend relationship as well as the references to the **extension points** are optionally shown in a **comment** note attached to the corresponding extend relationship.


Condition: {user clicked help link}
extension point: Registration Help
Registration
extension points
Registration Help
«extend»
Get Help On Registration

Registration use case is conditionally extended by Get Help On Registration use case in extension point Registration Help.

Example: Use Case description



Example 1 cont'd

Use Case: "Take Customer Order"

Basic Flow:

1. Actor enters Customer details
2. Actor enters code for product required
3. System displays Product details
4. Actor enters quantity required
5. Actor enters Payment details
6. System saves Customer Order

There are other Use Cases where we also need to record the Customer's details

Alternative Flows:

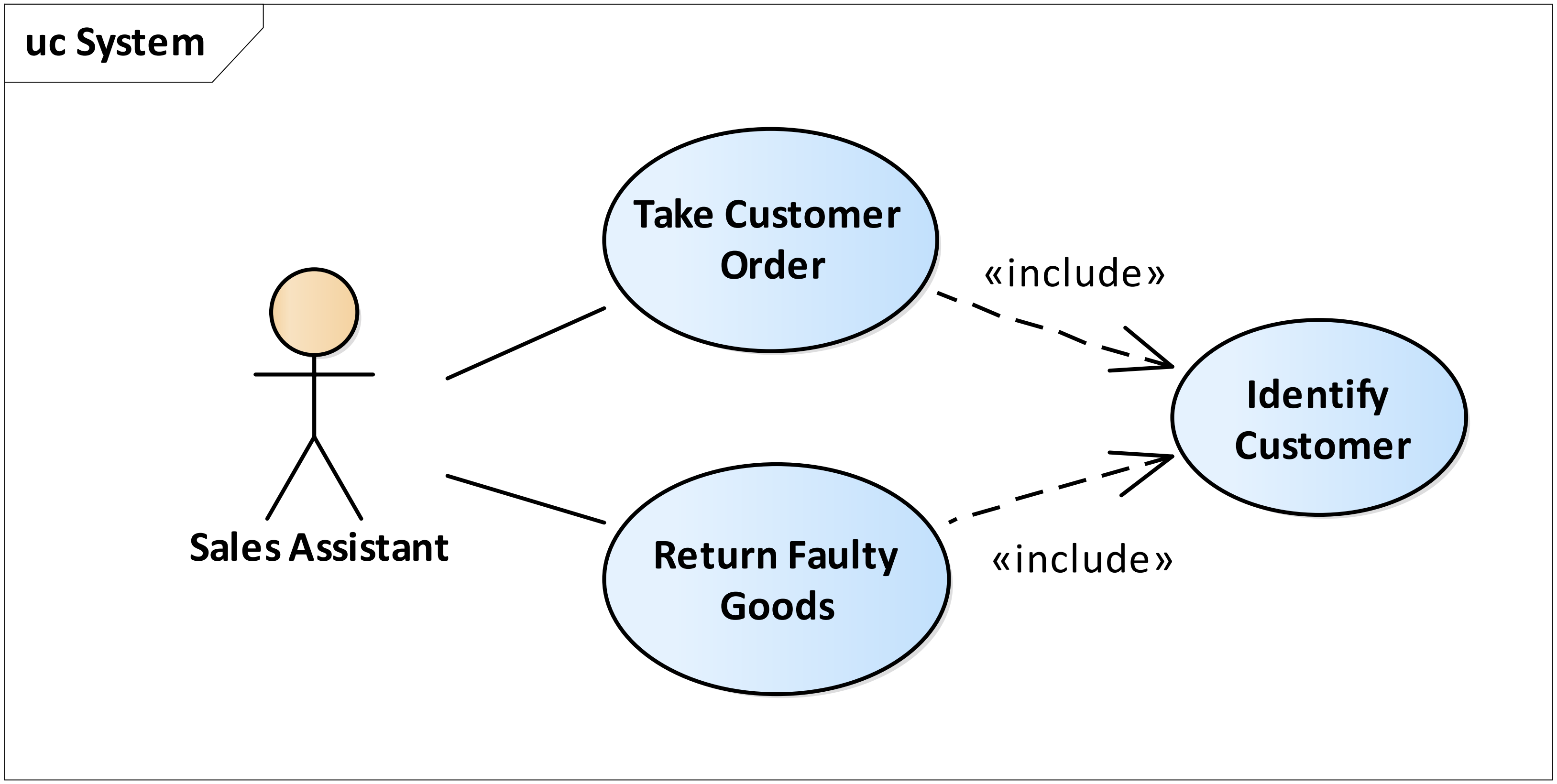
[multiple products]

After step 4, when the Actor enters the quantity required,

Repeat steps 2 to 4 for additional Products

Resume at step 5, to enter Payment details

Example: Include relationship



Example: Alternate flows

Example 2 – cont'd

Use Case: "Identify Customer"

Basic Flow:

1. Actor enters search criteria, surname and postcode
2. System displays matching Customers
3. Actor selects Customer
4. System displays Customer details
5. Actor confirms Customer

Alternative Flows:

[new customer]

After step 2, when the System does not display the required Customer, Actor creates new Customer,

1. Actor selects to add new Customer
2. Actor enters Customer details

Resume at step 5, to confirm Customer

Example: Specify the Include in the base use case

Example 2 – cont'd

Use Case: "Take Customer Order"

Basic Flow:

1. Actor records Customer details, **include** (Identify Customer)
2. Actor enters code for Product required
3. System displays Product details
4. Actor enters quantity required
5. Actor enters Payment details
6. System saves Customer Order

Alternative Flows:

[multiple products]

After step 4, when the Actor enters the quantity required,

Repeat steps 2 to 4 for additional Products

Resume at step 5, to enter Payment details

Example: Special treatment

- Suppose we want to sell products that are made to order and require a degree of customer specification.
- For these products, we will need to record the customer's additional requirements, such as size and color.
- In this case we are **adding something extra to the flow of the base Use Case.**
- We could do this as **an Alternative Flow**.

Example: Alternative flows

Example 3

Use Case: “Take Customer Order”

Basic Flow:

1. Actor records Customer details **include** (Identify Customer)
2. Actor enters code for product required
3. System displays product details
4. Actor enters quantity required
5. Actor enters payment details
6. System saves Customer Order

Alternative Flows:

[multiple products]

After step 4, when the Actor enters the quantity required,

Repeat steps 2 to 4 for additional products

Resume at step 5, to enter payment details

[customer specified product]

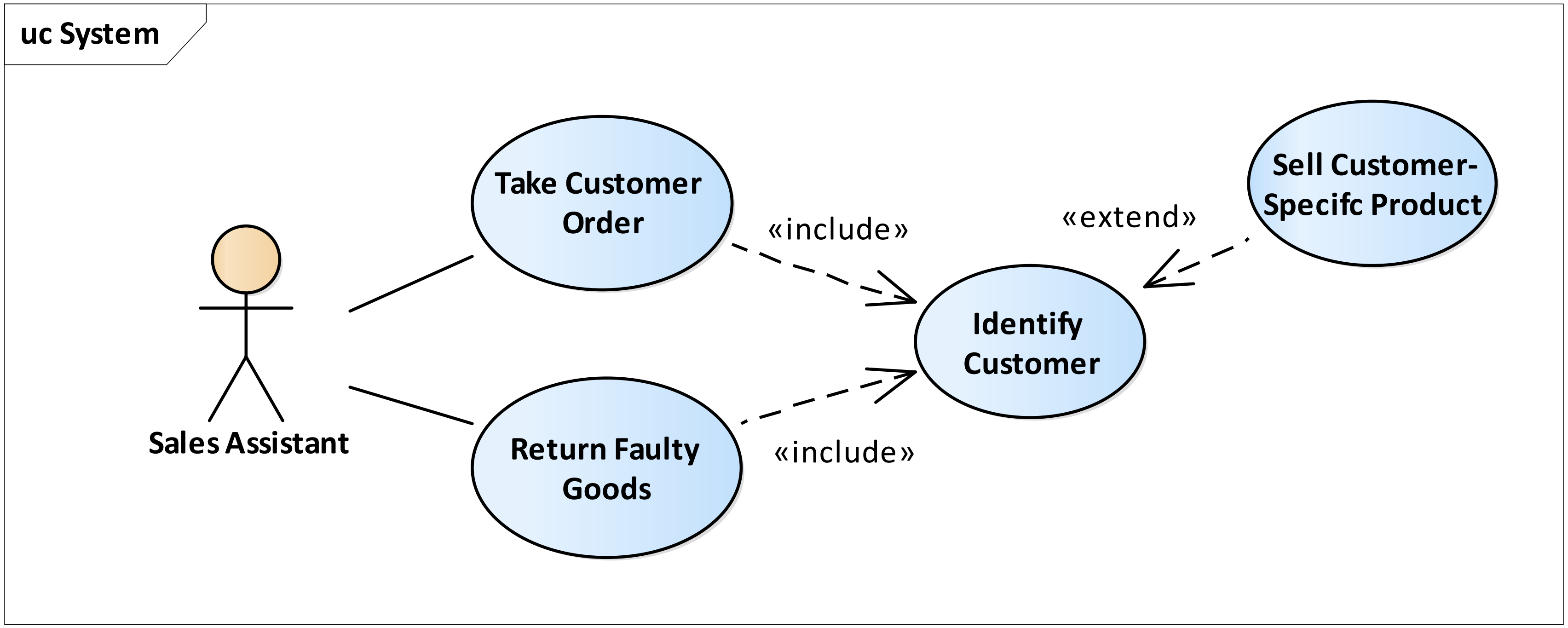
At step 3, when the System displays the Product details, if the product requires customer specified features,

1. Actor enters customer specified requirements, such as size and colour

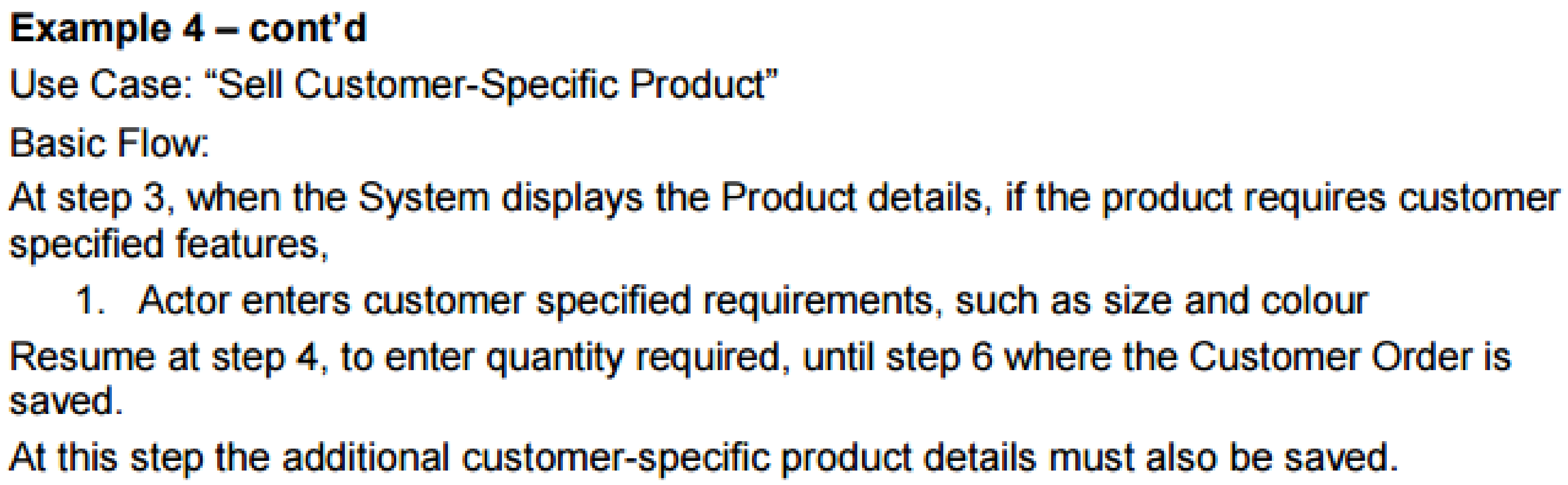
Resume at step 4, to enter quantity required, until step 6 where the Customer Order is saved.

At this step the additional customer-specific product details must also be saved.

Example: Extend relationship



Example: the extension



Example 4 – cont'd

Use Case: "Sell Customer-Specific Product"

Basic Flow:

At step 3, when the System displays the Product details, if the product requires customer specified features,

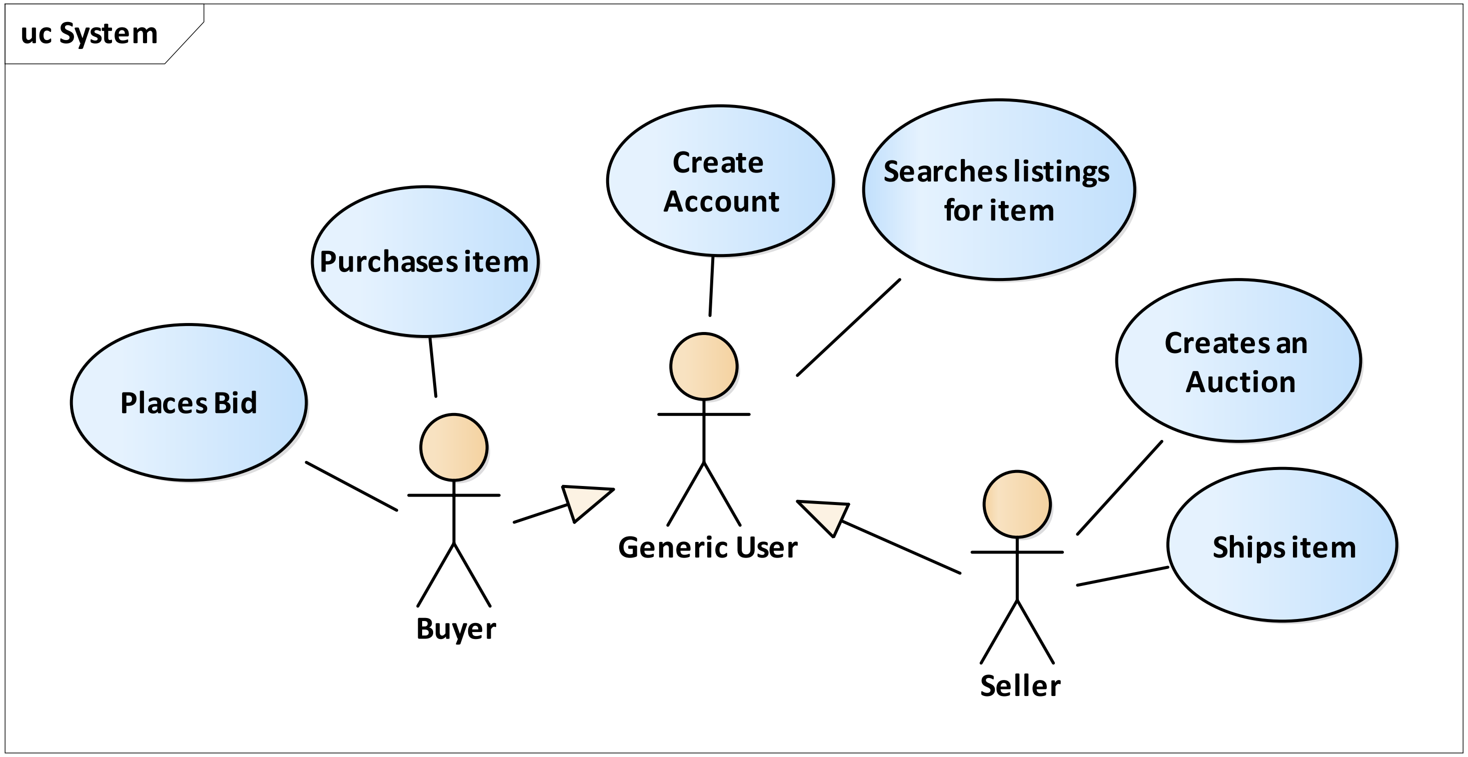
1. Actor enters customer specified requirements, such as size and colour

Resume at step 4, to enter quantity required, until step 6 where the Customer Order is saved.

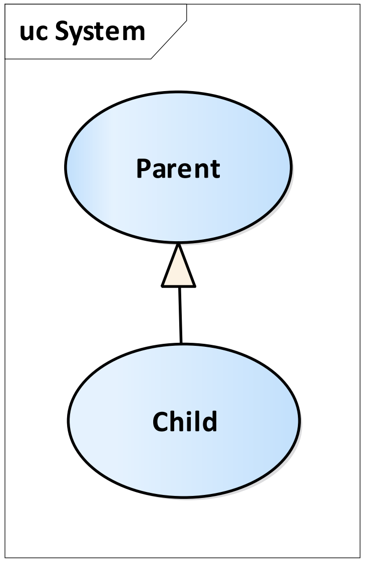
At this step the additional customer-specific product details must also be saved.

Reusing Use Cases through Actor Generalization

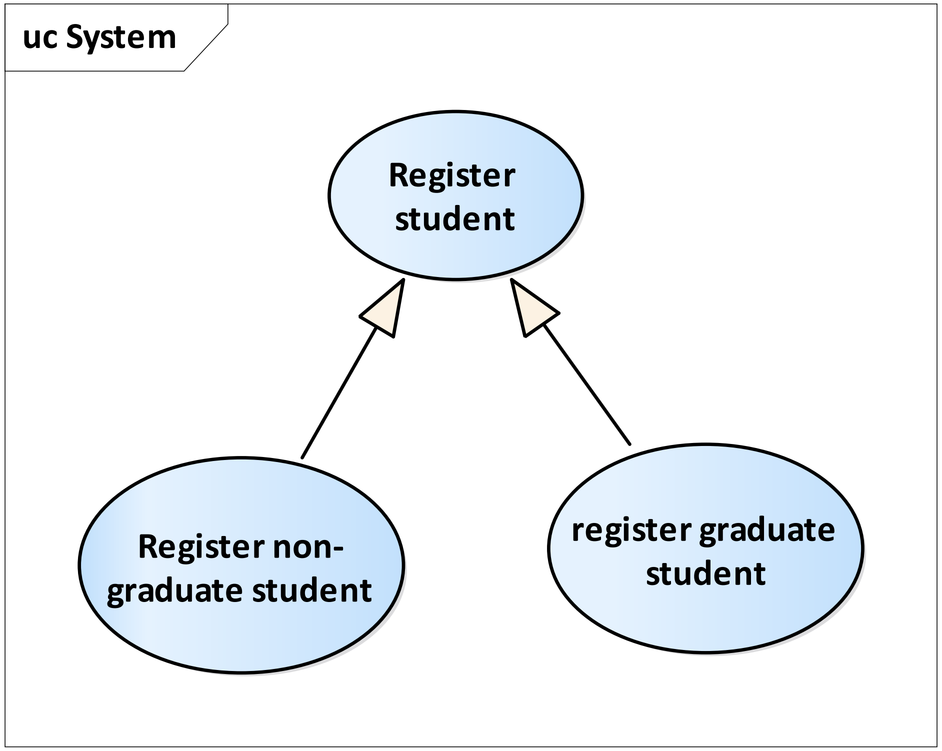
- There is **duplicate behavior** in both the buyer and seller which includes "create an account" and "search listings".
- Extract a **more general user** that has the **duplicate behavior** and then the actors will "**inherit**" this behavior from the new user.



Generalization of Use Cases

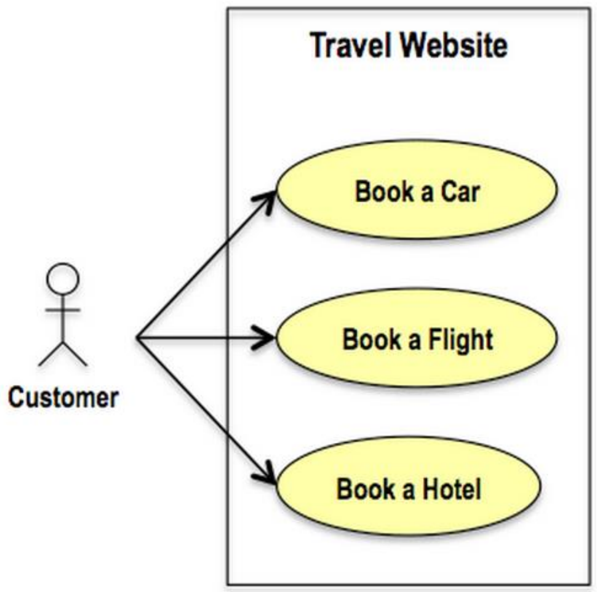


- The child use case **inherits** the behavior and meaning of the parent use case.
- The child may add to or override the behavior of its parent.

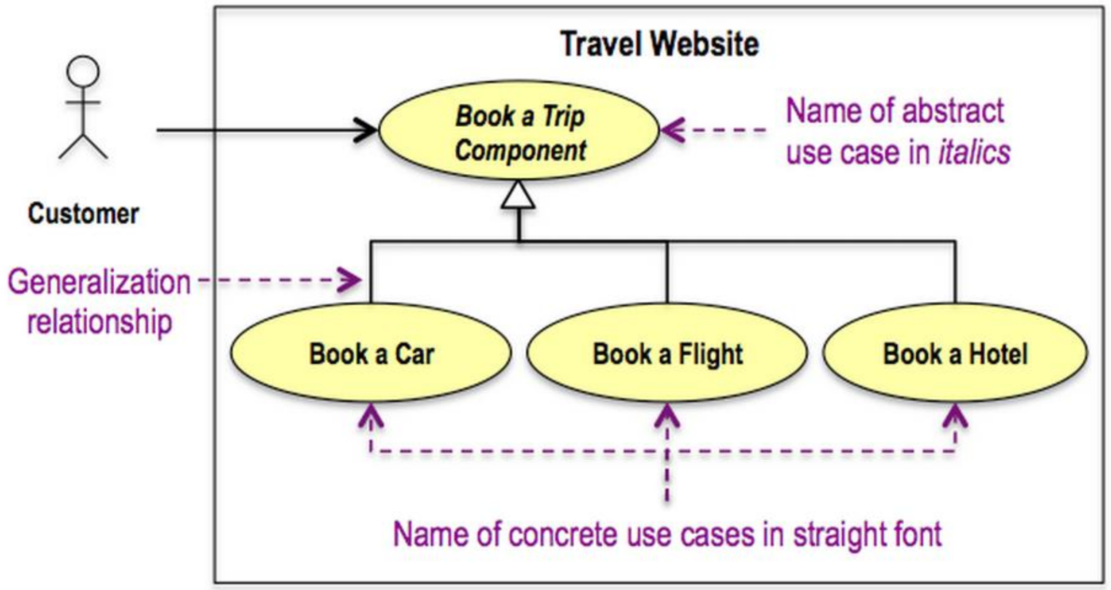


Generalization of Use Cases

Start: Model concrete use cases to represent different actor goals.



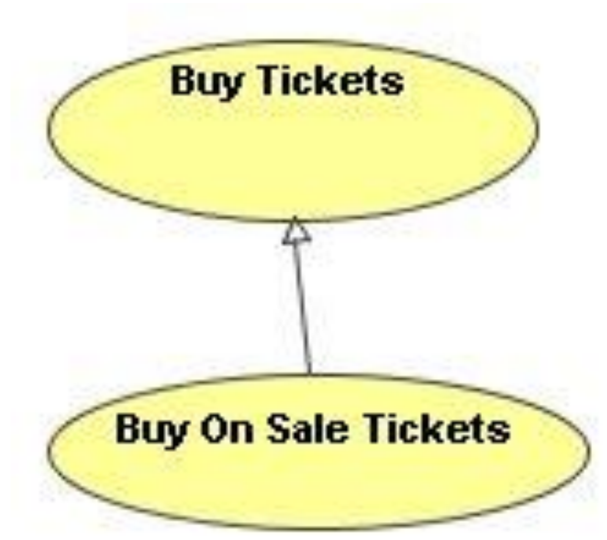
Optimize: Generalize the concrete use cases to a generalization use case to represent the core booking process. Generalize the associations to an association on the generalization use case, so the specialization use cases inherit it.


Travel Website
Customer
Book a Trip Component
Name of abstract use case in *italics*
Generalization relationship
Book a Car
Book a Flight
Book a Hotel
Name of concrete use cases in straight font

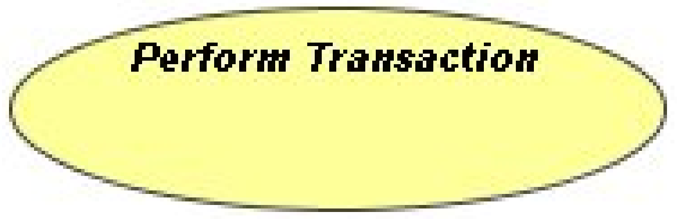
Note: From a Customer's point of view, nothing has changed. Any Customer still sees only the concrete (public) specialization use cases, not the abstract (private) generalization use case. This is about **model optimization** and nothing else.

Concrete vs. Abstract Use Cases

The "Buy Tickets" use case is **concrete** because it can be performed all by itself

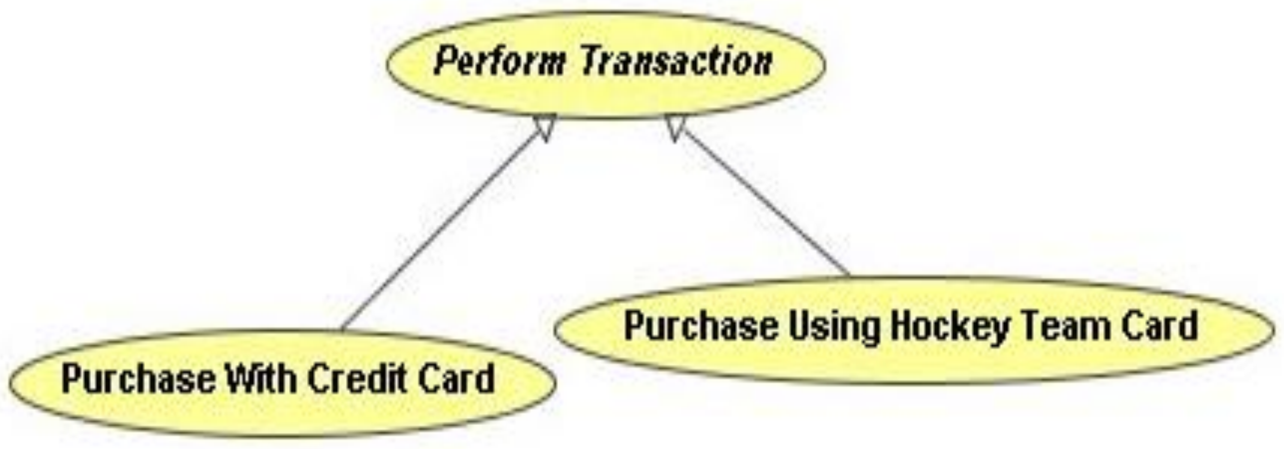


- Abstract use cases **cannot be performed**
- Abstract use cases only provide partial behavior and thus **they need to be implemented**
- Described as ***Italic***



Implementation Relationship

The generalization relationship is used to implement an abstract use case

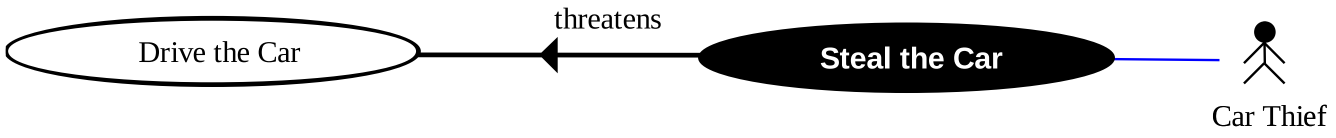


Misuse Scenarios/Cases

- Called also "misuse case" describes a **sequence of interactions in which a hostile actor uses the system against the stakeholders' intentions.**
- The execution of a misuse scenario represents **a threat for the system**, the **stakeholders, or other systems in the context.**

Example:
Tom, the driver of another car, intentionally cuts in right ahead of Carl in order to cause Carl's vehicle to perform a full braking. During this braking maneuver Carl is injured.
A hostile actor **knowingly** causes a **dangerous situation** and thereby **misuses the car safety system.**

Misuse Cases



- Models functional security requirements
- Valuable for hazard and threat analysis
- Misuse cases are negative use cases
- Actor is a **hostile agent,** called also **mis-actor**
- Extension of use case modeling
- Used for test cases generation

Misuse Cases

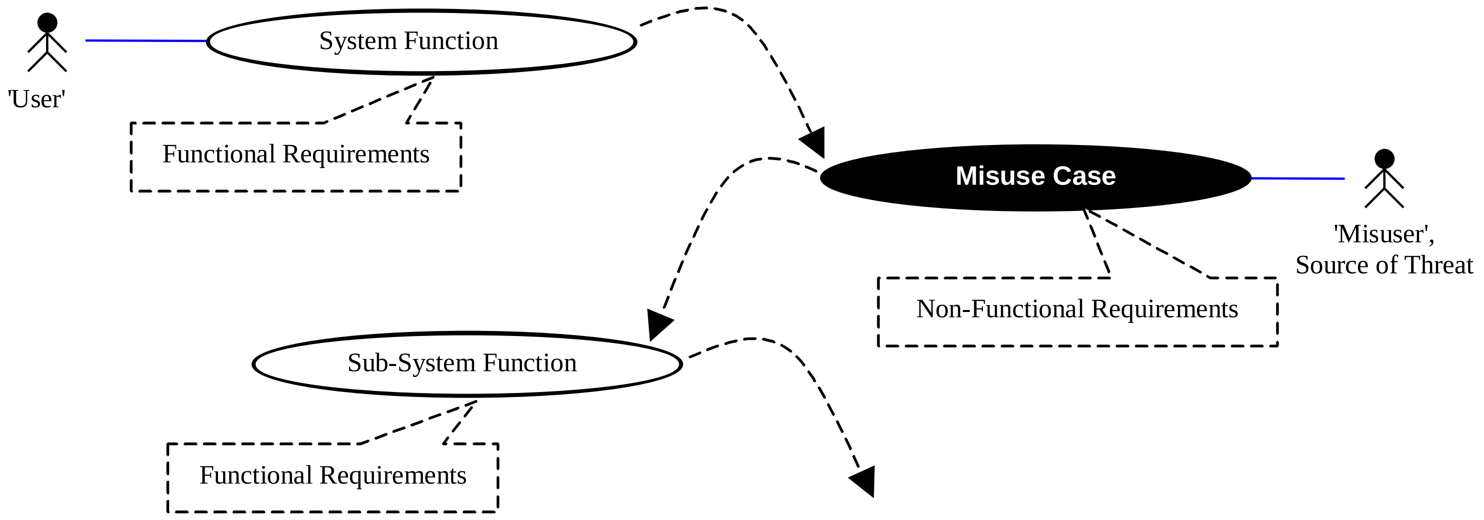
- A misuse case model consists of:
  - Misuse case diagram
  - Misuse case descriptions

- The misuse case model makes use of **include**, **extend**, **generalize** and **association**.

Two new relations to be used in the diagram:

- **Mitigates:** A use case can mitigate the chance that a misuse case will complete successfully.
- **Threatens:** A misuse case can threaten a use case, e.g. by exploiting it or hinder it to achieve its goals.

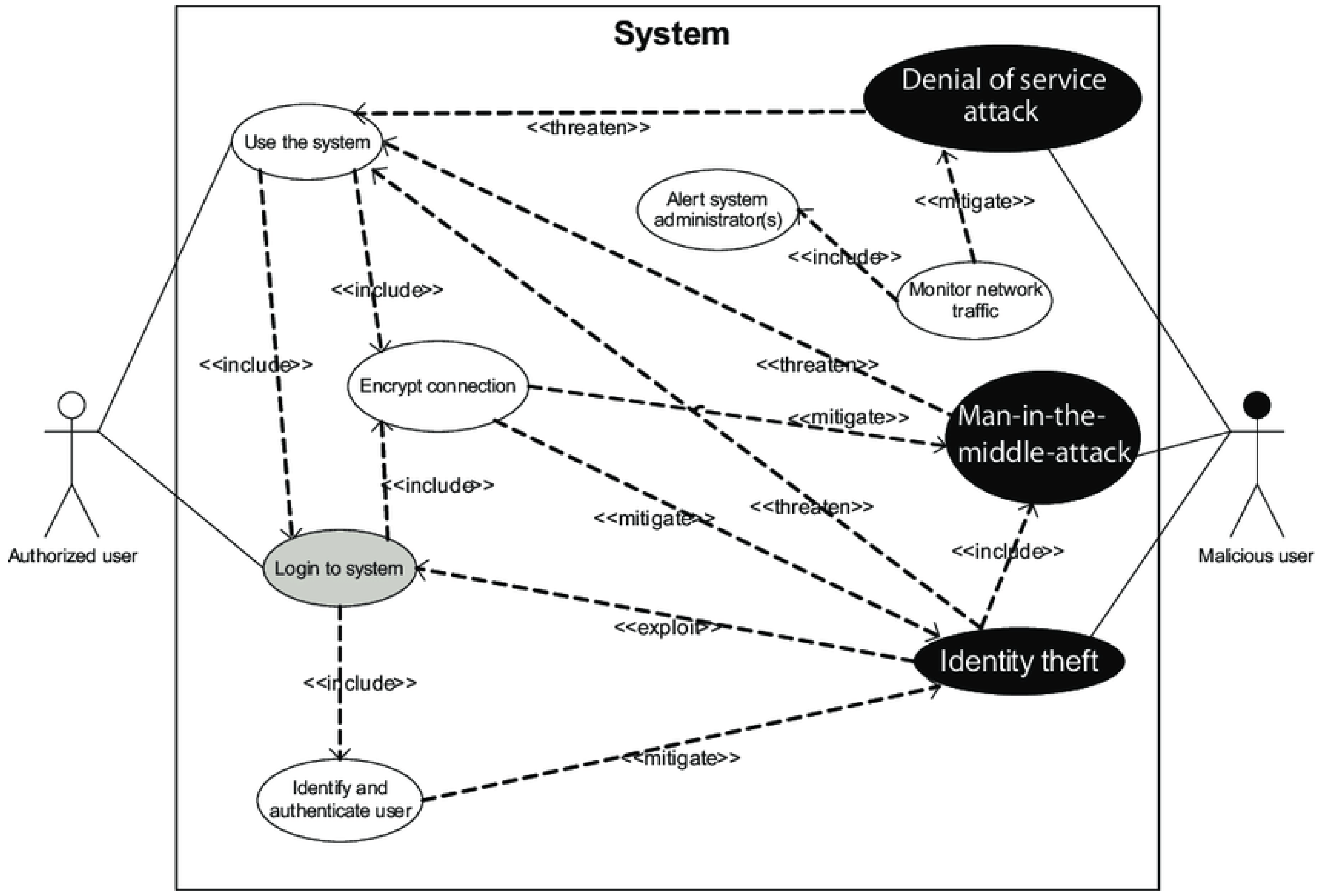
Misuse Cases Identify NFRs



Interplay of Use & Misuse Cases with Functional & Non-Functional Requirements

- Use Cases are weak on NFRs
- Misuse Cases naturally focus on NFRs, e.g. Safety, Security
- Response is often a subsystem function, possibly to handle an Exception

Misuse Case Diagram



Misuse Cases Documentation

There are two ways to textually describe a misuse case:

1. **Embedded in a use case description template** - where you add an extra description field called **Threats**. This is the field where you fill in your misuse case steps (and alternative flows). **This is referred to as the *lightweight* mode of describing a misuse case.**

2. **Use of a separate template**. Inherit some of the fields from use case description (**Name**, **Summary**, **Author** and **Date**). It also adapts the fields **Basic path** and **Alternative path**, where they now describe the paths of the misuse cases instead of the use cases.

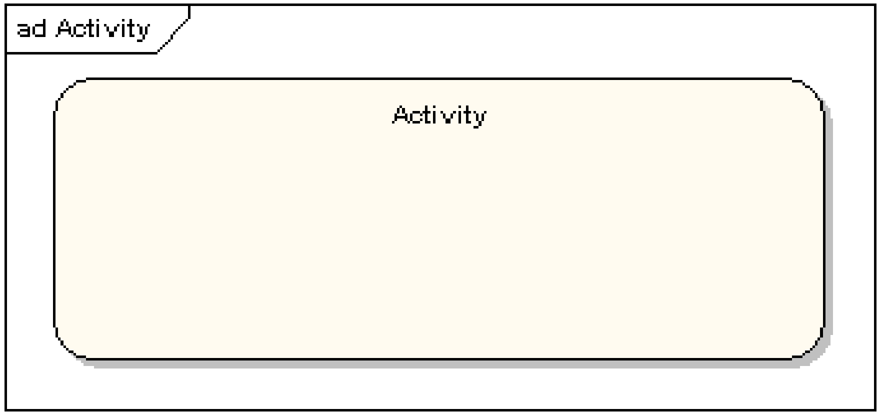
UML Activity Diagram

Activity Diagrams

- Model the flow of activity/events **from a start point** to **the finish point** detailing the many **decision paths** that exist in the progression of activities/events contained in the activity.
- May be used to detail situations where the **logic is complex** and there are **a lot of alternate flows** (e.g., **parallel processing** may occur in the execution of some activities).
- Typically used for **business process modeling**, for modeling the logic captured by a single use case or usage scenario, or for modeling the detailed logic of a business rule.
- If customers prefer diagrams over text.

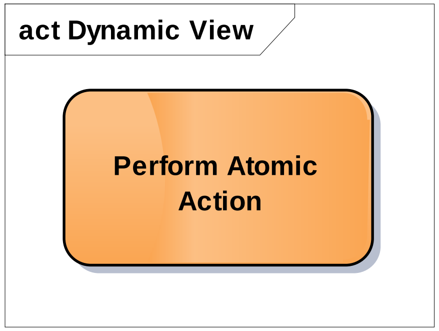
Activity

An activity is shown as a round-cornered rectangle **enclosing** all the **actions**, **control flows** and **other elements** that make up the activity.



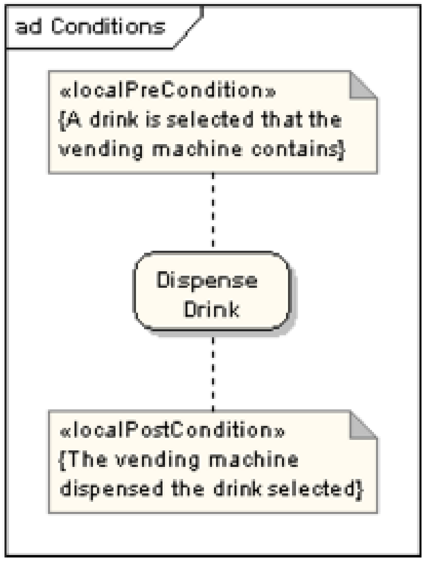
Actions

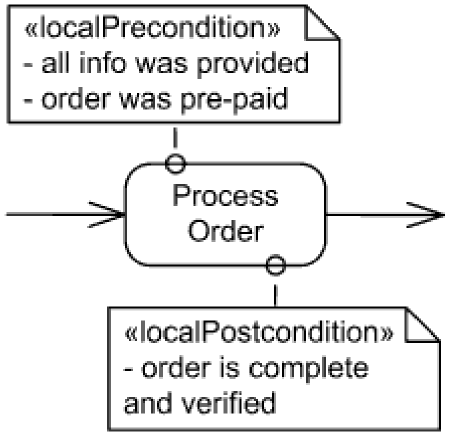
- An action represents a **single step** within an activity (**one that is not further decomposed within the activity**).
- In Enterprise Architect (EA), it is referred to as "**Atomic**" action.
- Actions are denoted by **round-cornered rectangles**.



Action Constraints

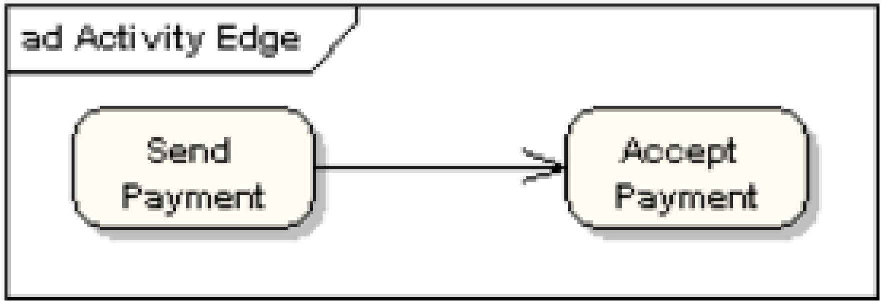
Constraints can be attached to an action (e.g., **local pre- and post-conditions**).


ad Conditions
«localPreCondition» {A drink is selected that the vending machine contains}
Dispense Drink
«localPostCondition» {The vending machine dispensed the drink selected}

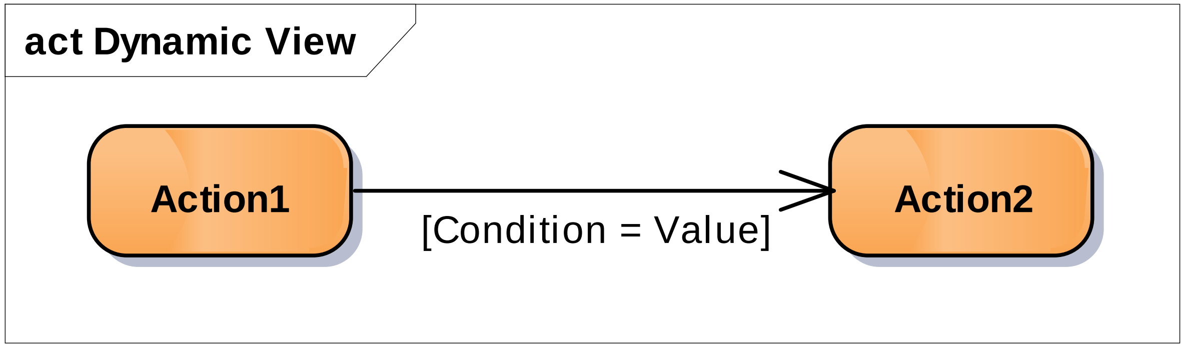

«localPrecondition» - all info was provided - order was pre-paid
Process Order
«localPostcondition» - order is complete and verified

Control Flow

A control flow shows the flow of control from one action to the next. Its notation is a line with an arrowhead.



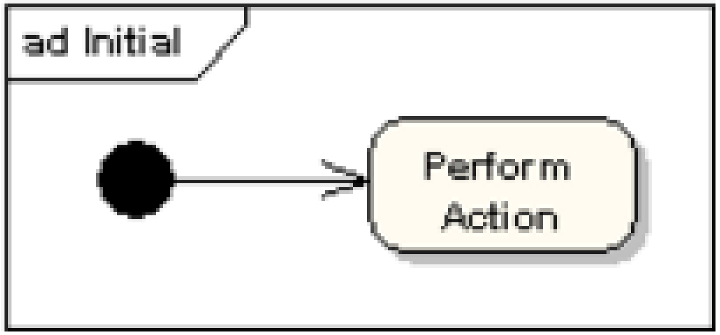
The control flow may have a condition attached to it.



Initial Node

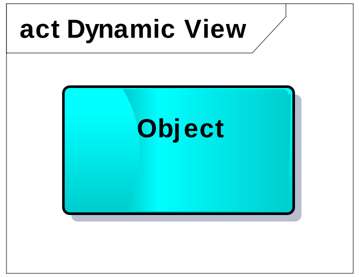
An initial or start node is depicted by a large black spot.

You can have more than one initial node.

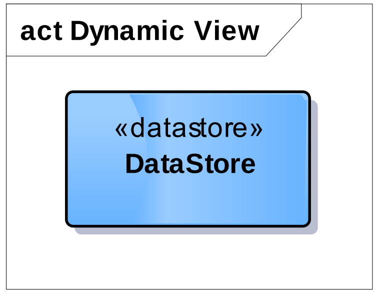


Objects

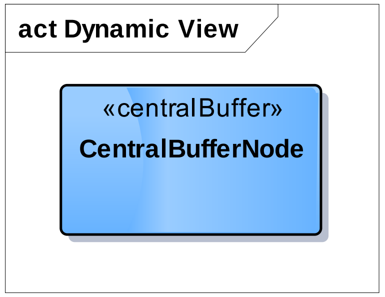
An object is shown as a rectangle.



- A *Datastore* is a **persistent** buffer node. A data store is shown as an object with the «datastore» keyword.

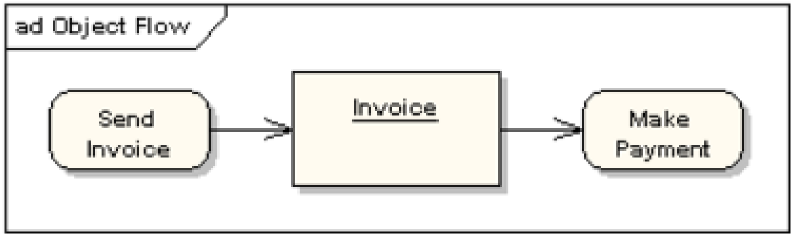


- A *Central Buffer Node* is a **transient** buffer node. It has the same behavior as a Datastore, but the stored content will be destroyed when the activity ends – when an Activity Final is reached.

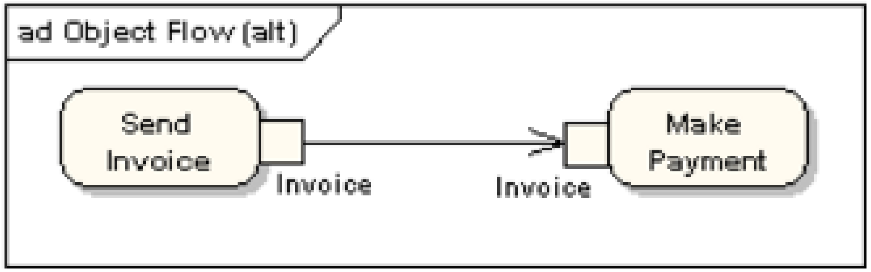


Object Flows

- An object flow is a path along which objects or data can pass.
- An object flow is shown as a connector with an arrowhead denoting the direction the object is being passed.

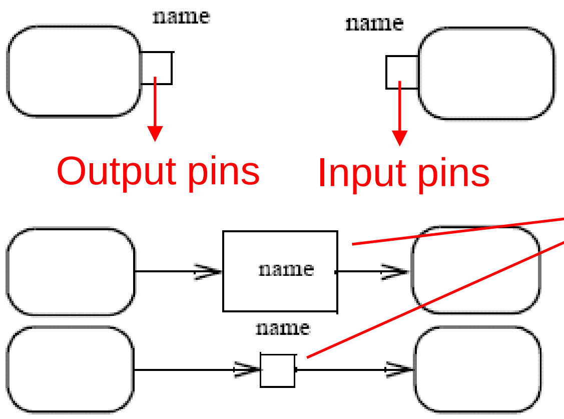


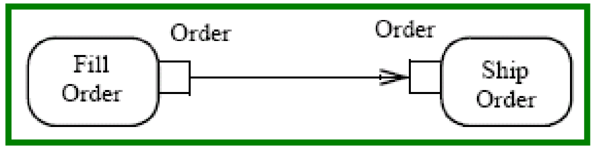
- An object flow must have an object on at least one of its ends.
- A shorthand notation for the above diagram would be to use input and output pins.



Pins

- Actions can have **inputs** and **outputs**, through the pins
- Hold inputs to actions until the action starts, and hold the outputs of actions before the values move downstream
- The name of a pin is not restricted: generally recalls the type of objects or data that flow through the pin

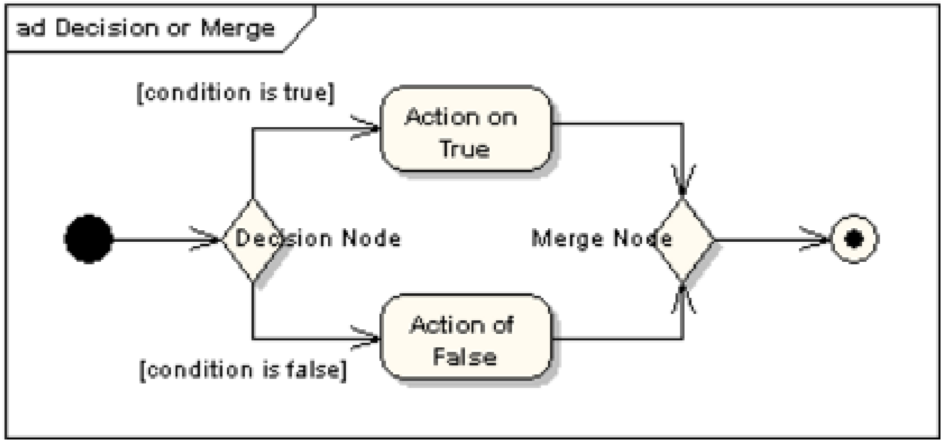




Standalone pin notations: the output pin and the input pin have the same name and same type

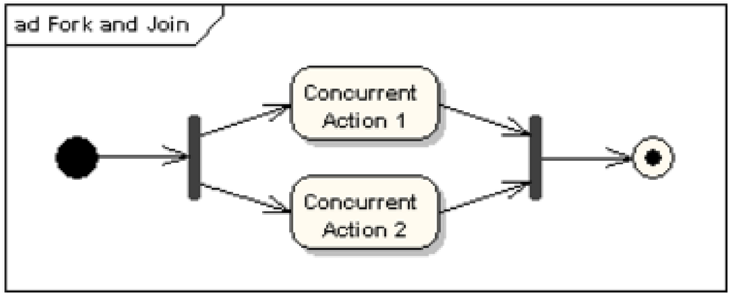
Decision and Merge Nodes

- Decision nodes and merge nodes have the same notation: a **diamond** shape.
- They can both be named.
- The control flows coming away from a decision node will have guard conditions which will allow control to flow if the guard condition is met.



Fork and Join Nodes

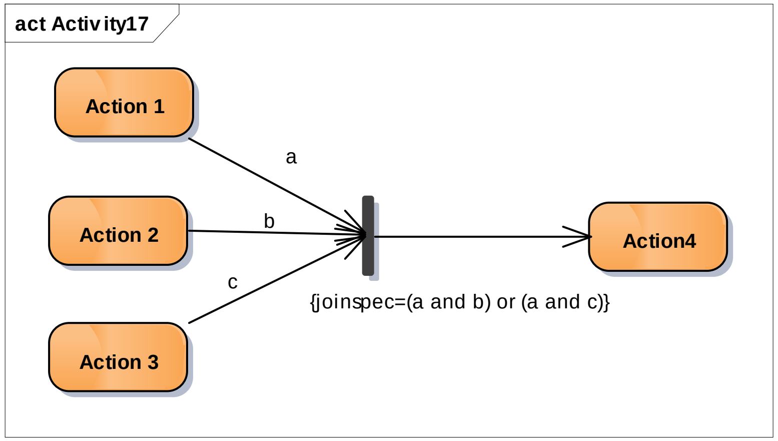
- Forks and joins have the same notation: either a horizontal or vertical bar. (the orientation is dependent on whether the control flow is running left to right or top to bottom).
- **They indicate the start and end of concurrent threads of control**.
- A join node may have two or more incoming legs. For continuation it's necessary that all reach the join node.



Note: In Enterprise Architect, the type join/fork is selected in Properties → kind

Join Specification Feature

If only some of the arriving tokens shall be sufficient to continue with the synchronized path, UML provides the *Join Specification* (JoinSpec) feature. By this you may specify a condition, sufficient for synchronization.



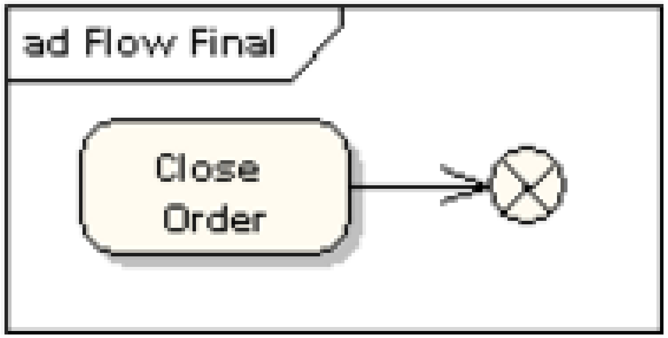
Note:

In Enterprise Architect, the type join/fork is selected in: Properties → joinSpec

Merge vs. Join Nodes

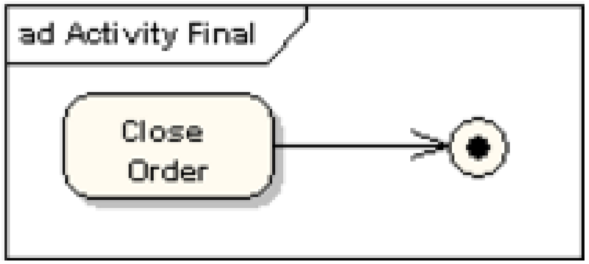
- A join is different from a merge in that the join synchronizes two inflows and produces a single outflow.
- The outflow from a join cannot execute until all inflows have been received.
- A merge passes any control flows straight through it.
- If two or more inflows are received by a merge symbol, the action pointed to by its outflow is executed two or more times.

Flow Final Node



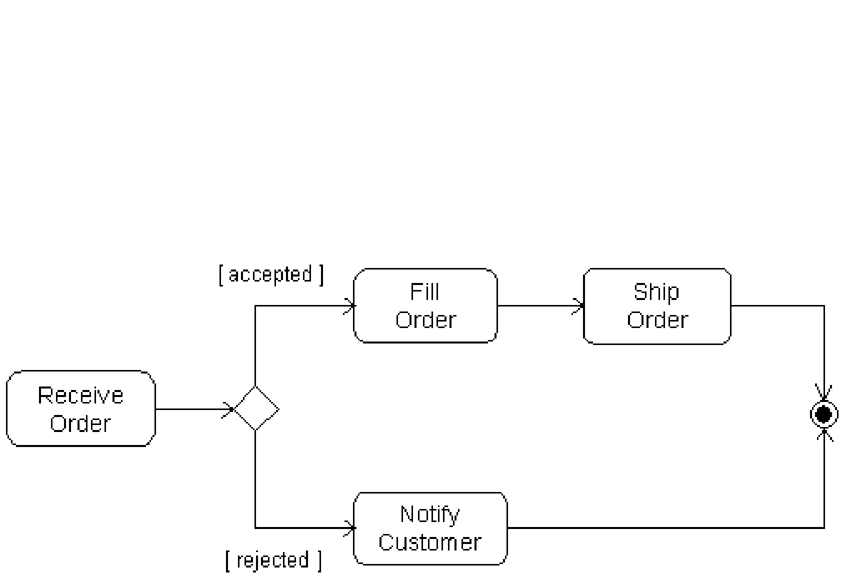
- Depicted as a circle with a cross inside
- Denotes the end of a **single control flow**
- A flow final destroys all tokens that arrive at it. **It has no effect on other flows in the activity.**
- **You can have more than one flow final node.**

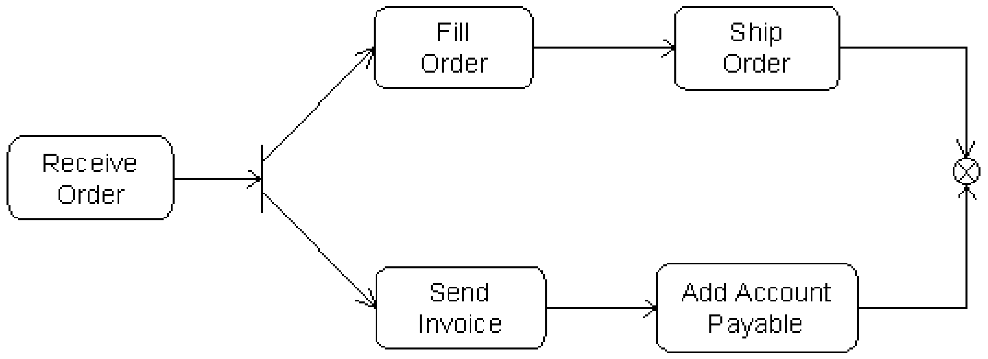
Activity Final Node

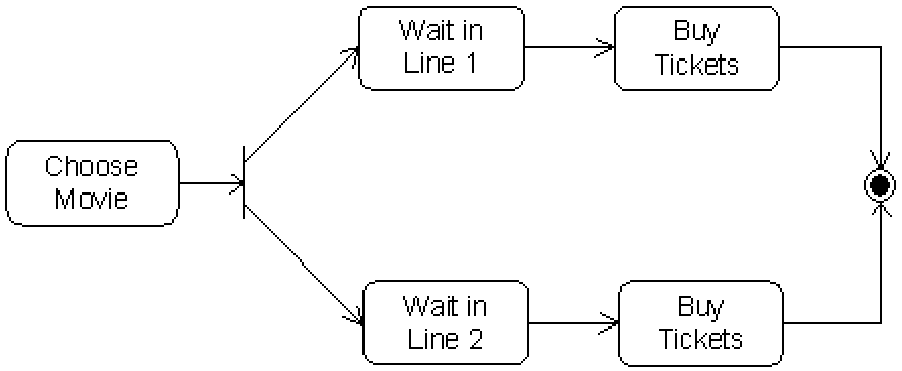


- Denotes the end of all control flows within the activity.
- Depicted as a circle with a dot inside.
- An activity may have **more than one activity final node**. The first one reached **stops all flows** in the activity.

Flow Final vs. Activity Final

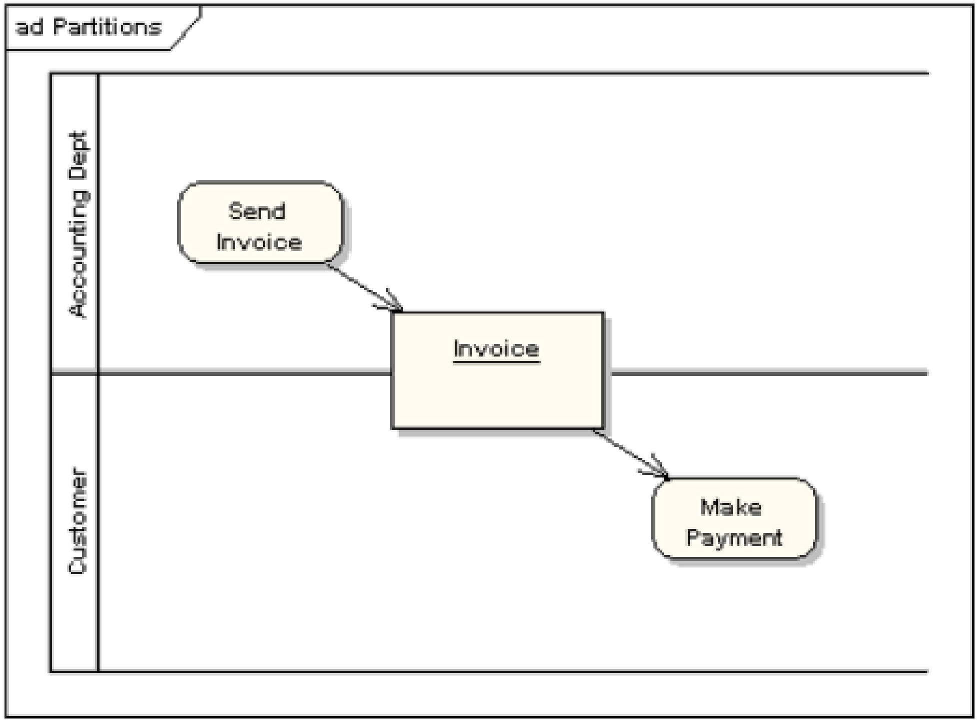




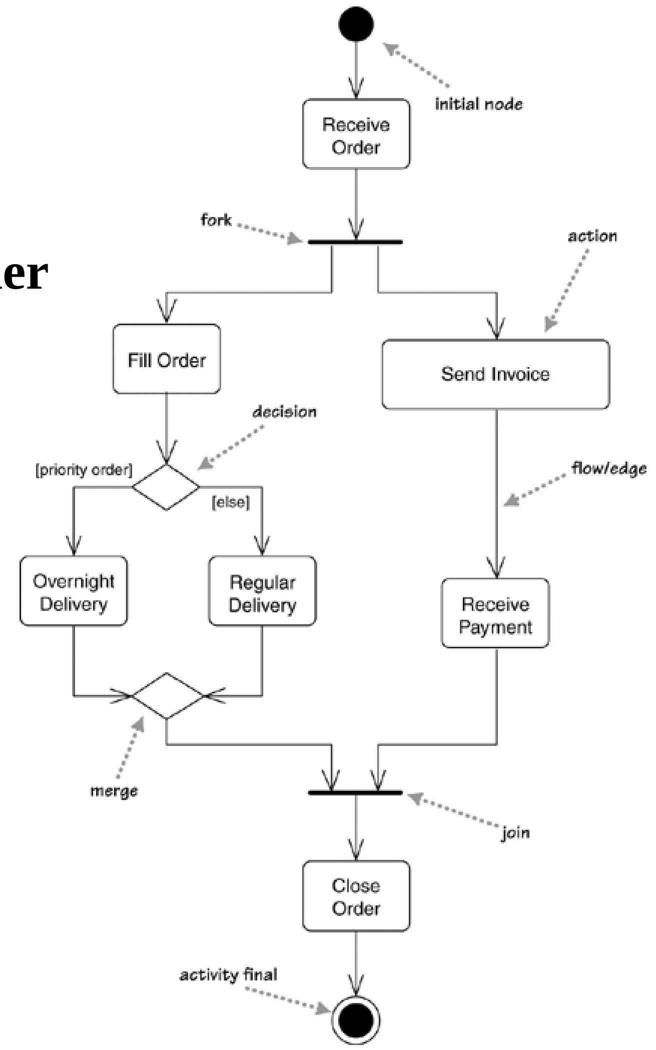


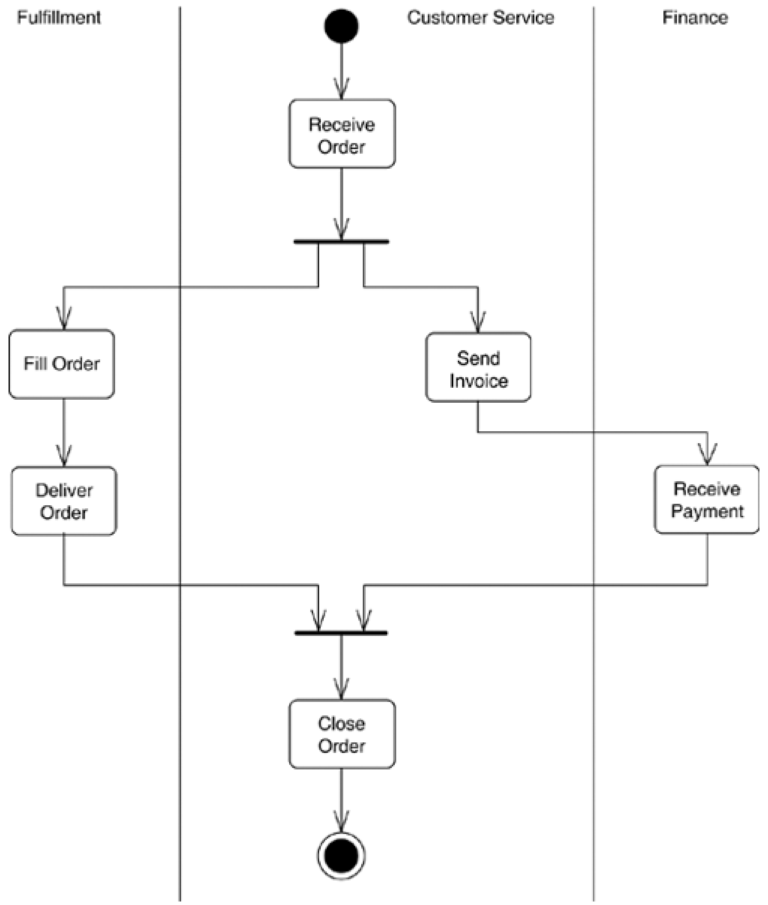
Partitions

- An activity partition is shown as either **a horizontal or vertical swimlane**.
- The partitions are used to **separate actions** within an activity into those performed by the accounting department and those performed by the customer.



Process Order


initial node
Receive Order
fork
action
Fill Order
Send Invoice
decision
[priority order]
[else]
flow/edge
Overnight Delivery
Regular Delivery
Receive Payment
merge
join
Close Order
activity final


Fulfillment
Customer Service
Finance
Receive Order
Fill Order
Send Invoice
Deliver Order
Receive Payment
Close Order