Object Oriented Analysis and Design

Use Case Diagram

Ridwan Rismanto

System requirement

Use case diagrams are a very powerful tool in UML

- Describe the interaction between the user and the system
- Describe the entire system to be made

Before making a use case, the system requirements, or requirements, or features that are in the system to be created must first be described.

Use case

Use cases are described with verbs.

- "Pay Bills", "Update Payroll", "Create Account"

In the use case diagram there is also a description of each interaction between the user and the system.

Use case notation:

Actor

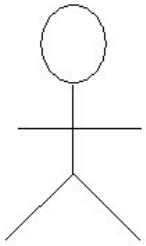
The use case cannot stand alone if there are no **actors**

Actor is *something* that can interact with use case

Example in the banking system

- There is a use case "Withdraw Money"
- Then there should be “customer” actors who can withdraw money (take money)

Actor notation:



Customer

Actor is not always human

An actor can also be something outside the system that interacts with the use case.

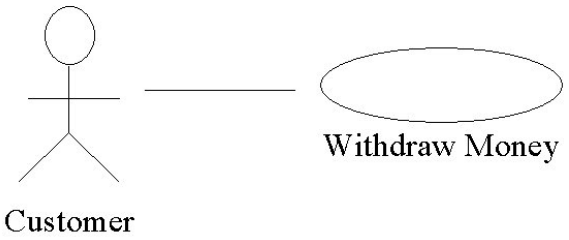
- For example, other systems that interact with the system to be created.

An actor can also be an abstract concept, such as a **time** , or a **date** .

- For example, there is a use case "Delete expired orders" in the information system for ordering goods, and actors who interact with the use case are "Holidays"

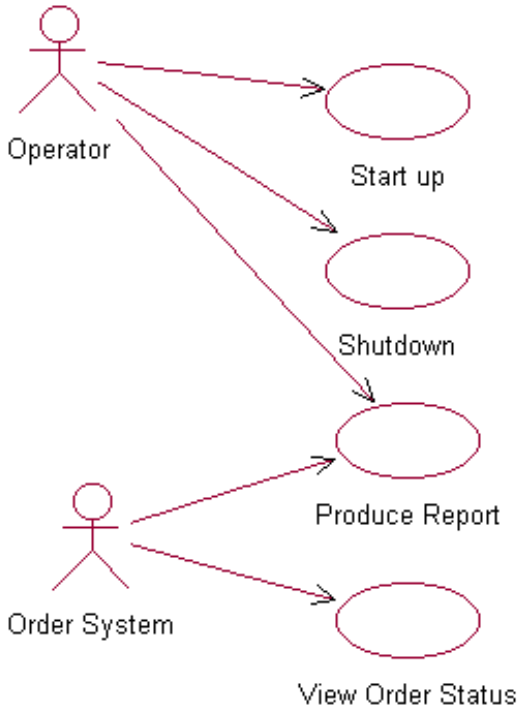
Interaksi actor dengan use case

The interaction between actors and use cases is described in straight lines:



An actor can interact with

many use cases, and vice versa:



Purpose of the use case diagram

Defines the scope of the system

- Visualize the size and scope of the entire development process

Use cases are very similar to system requirements

- However, it is more detailed and focused

Use case is a description of the whole system

- That is, something that is not in the use case is not included in the system to be created

Allows communication between user and developer

- Because the diagram is simple and easy to understand

Use cases can be used as references by the development team

- Or the backbone of the development process

Can easily do planning in the develop process

Can be used as a basis for testing and testing the system

Can assist in making manuals to use the system or *user guides*

Determining use cases

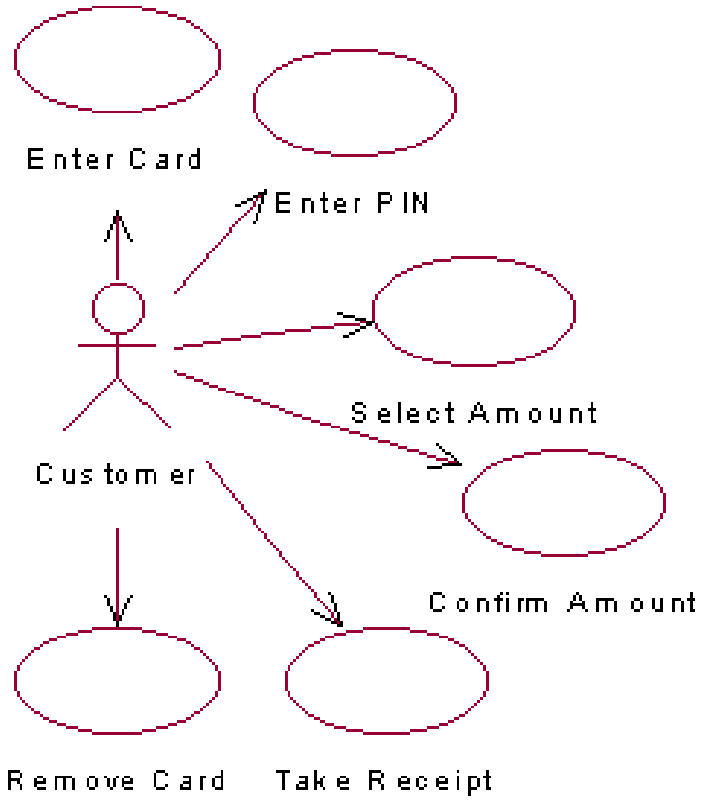
Errors that often occur in making use cases:

For example, an ATM machine that has a feature to withdraw money. The scenario:

- Insert an ATM card
- Enter the PIN
- Select the amount of money to be withdrawn
- Confirm withdrawal of money
- Take an ATM card
- Take the receipt

If we translate each of the above interactions into a use case, the result will be something like this:

Determining use cases (continued)



Determining use cases (continued)

The use case above seems fine.

But if that is the case, then in a complex system, the use case will become bloated and too redundant.

The key to making a use case:

The use case must fulfill the actor's goals

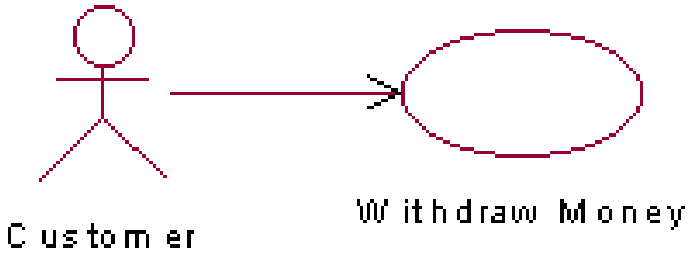
In this case, is "Receipt receipt" included in the purpose of ATM machine users?

- Will the world end if the receipt doesn't come out?

Based on this, the actual main goal of the actor is to "withdraw money" or take money.

So we should create use case as follow:

Determining use cases (continued)

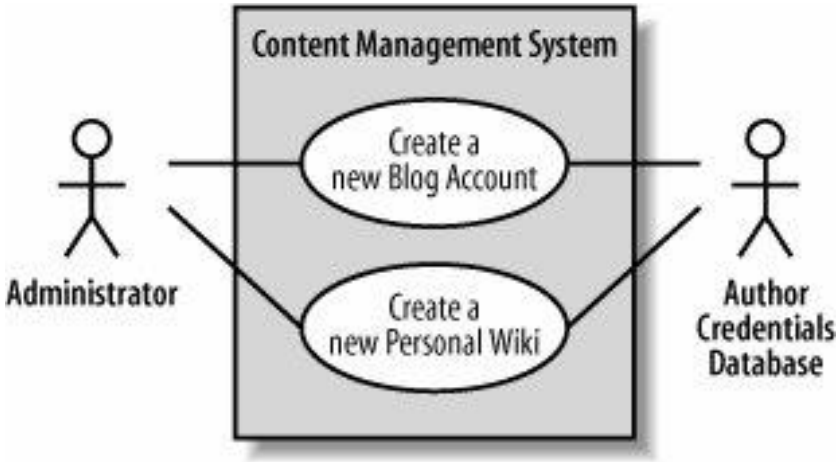


System boundaries

Used to describe the boundaries of a system.

Described by a box, with the name of the system, and inside is the use case. However actors are depicted outside the box.

Example:



Relationships in use case

Relationships in use cases consist of several types:

- Generalization
- Extend
- Include

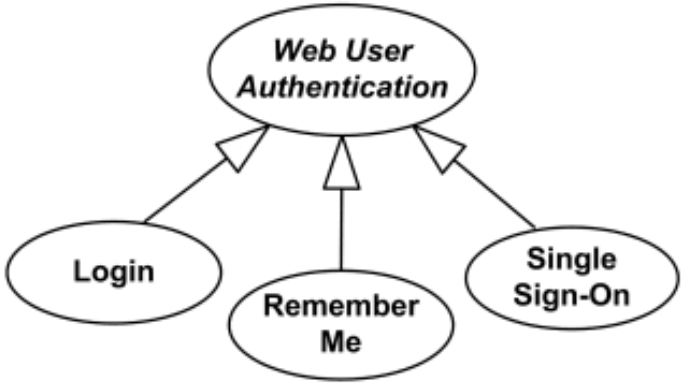
Relationship: Generalization

Use case generalization relationships are similar to *inheritance.*

Used when there is a use case whose main purpose is the same, but more specific.

Depicted by an arrow pointing to the parent use case.

For example, in the use case "web user authentication", there are several different methods, but the end goal is the same, namely user authentication:



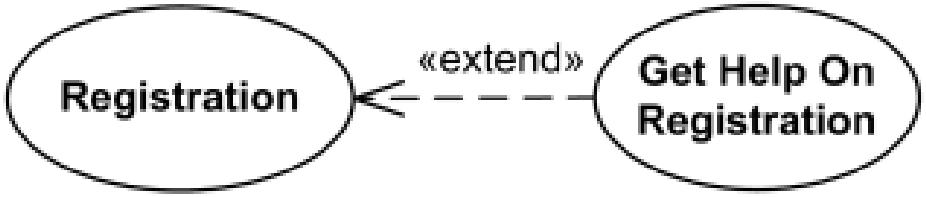
Relationship: Extend

Extend relation, is if there is a use case that can call another use case, but it is optional.

Depicted by a dashed arrow, with the caption <<extend>>, pointing **from an optional use case to the main use case** .

For example, in a “registration” use case, there is an optional use case “get help on registration” which is optional.

So the main use case is "registration", and optionally or extend it is "get help on registration".

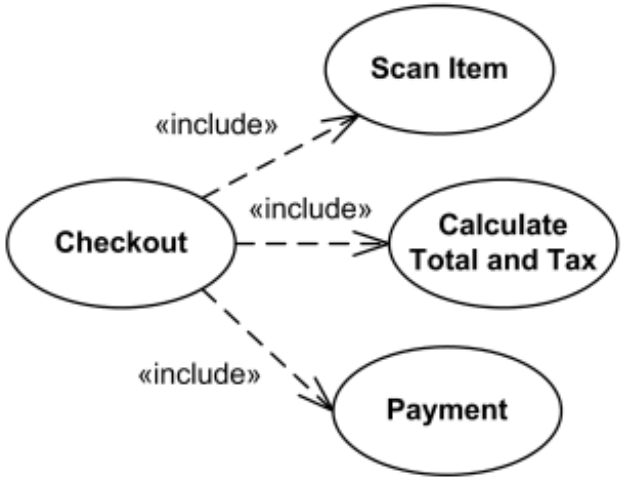


Relationship: Include

Relationships include is if a use case requires a call to the other and its use case **should** or *mandatory* .

Depicted by a dashed arrow, with the caption <<include>>, pointing from the **main use case to the use case included**

For example in the use case "checkout" on a supermarket cashier system, it must involve the use case "scan item", "calculate total and tax" and "payment". Then:



Use case description

A use case is always accompanied by a detailed description.

Use case description template:

Use Case:	Use case name
Deskripsi:	Brief description of the use case
Pre-conditions:	Certain conditions / conditions for this use case to run
Post-conditions:	Conditions after this use case is execute
Main flow:	What interactions are carried out to run this use case, for example for the use case "withdraw money" then the main flow is "insert card, insert pin, etc ..."
Alternate flow:	If there is an alternative interaction
Exception flow:	If there is a scenario to handle unexpected events, for example if the balance is not enough, etc ...

Use case in the analysis phase

Identify as many use cases as possible, but don't need to be too detailed.

- Remember the *key to making use cases*

After identifying existing use cases, cross references can be made with the system requirements.

There is no need to force something to become a use case or actor.

- Usually additional use cases will appear in the further design process later. And this is normal.

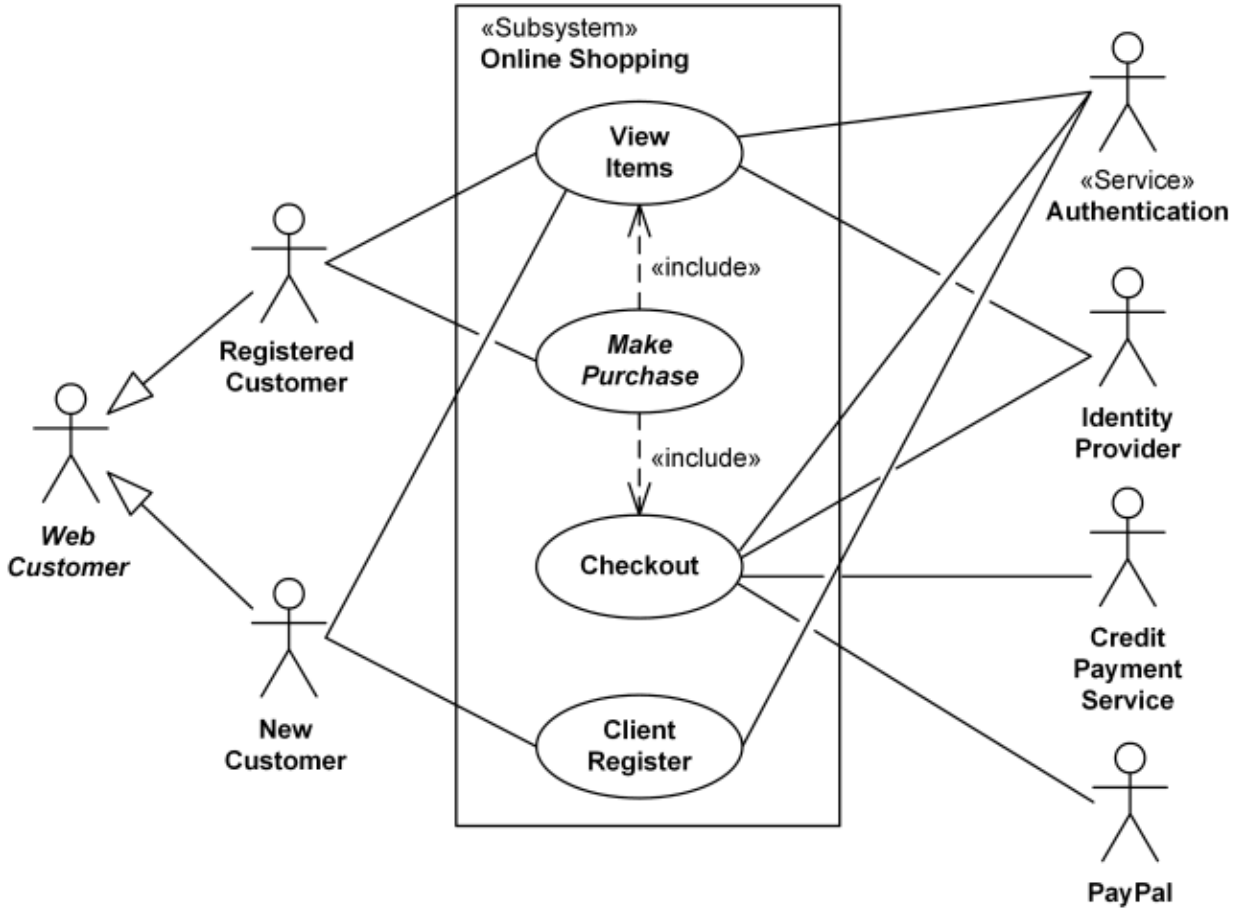
Case study: online shopping

Requirements:

- Web-based application with user authentication.
- Customers who have registered can view items and make purchases and payments.
- Customers who have not registered can view items and register.
- There are facilities:
  - Search for items
  - Browse items
  - Recommended item
  - Shopping cart
  - Wishlist

Case study: online shopping

Top level use case:



Case study: online shopping

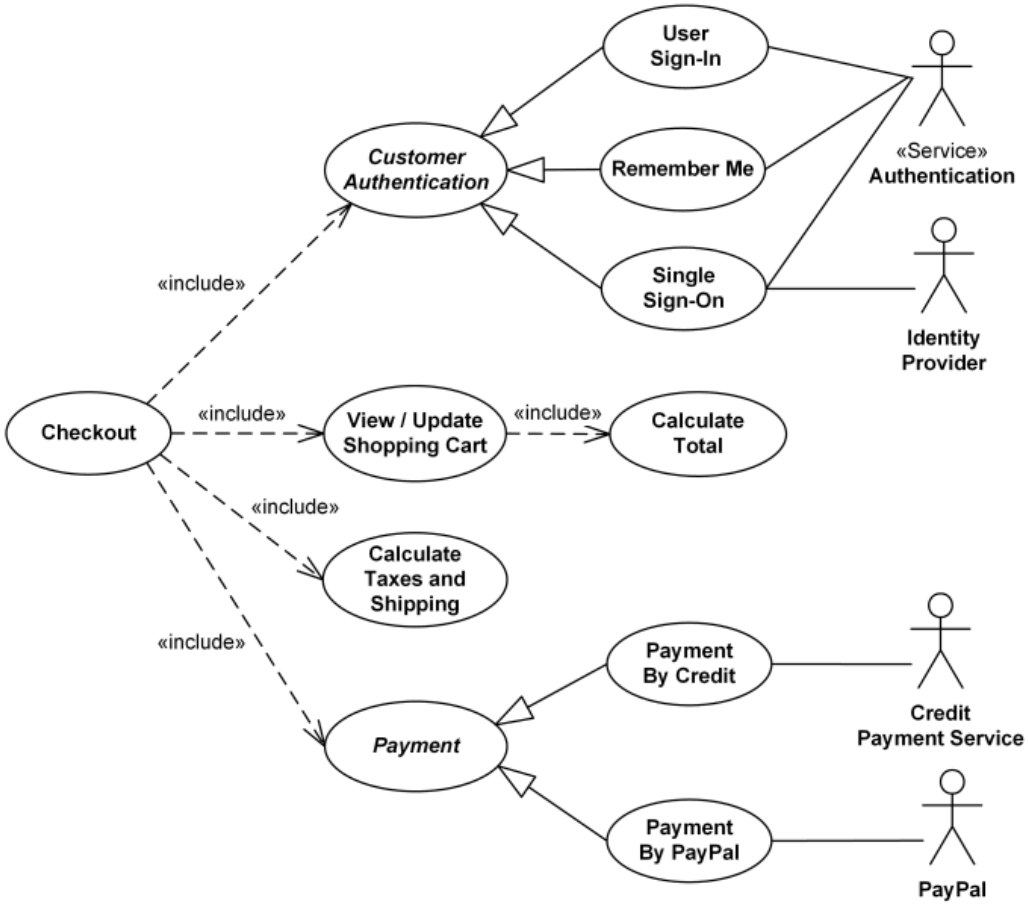
The "view items" use case can be extended with other use cases that are optional.

The use case "customer authentication" is included with the use case "view recommended items" and "add to wishlist" because the customer must be logged in.

But on the other hand, to add items to the shopping cart, customers don't need to log in.

Case study: online shopping

The use case "checkout" is included with other use cases that are mandatory.



Exercise

Create system requirements and use case diagrams (choose one):

- Online airline reservation
- Online hotel reservation

Thanks!

Any questions?

You can find me at:
rismanto@polinema.ac.id