RELATING USE CASES

Use cases can be related to each other. For example, a sub function use case such as Handle Credit Payment may be part of several regular use cases, such as Process Sale and Process Rental. It is simply an organization mechanism to (ideally) improve communication and comprehension of the use cases, reduce duplication of text, and improve management of the use case documents.

Kinds of relationships

1. The include Relationship
2. The extend Relationship
3. The generalize Relationship

1. The include Relationship

It is common to have some partial behavior that is common across several use cases. For example, the description of paying by credit occurs in several use cases, including Process Sale, Process Rental, Contribute to Lay-away Plan.. Rather than duplicate this text, it is desirable to separate it into its own subfunction use case, and indicate its inclusion.**Use cases and use the include relationship when:**

- They are duplicated in other use cases.

- A use case is very complex and long, and separating it into subunits aids comprehension.

Concrete, Abstract, Base, and Addition Use Cases

A **concrete use case** is initiated by an actor and performs the entire behavior desired by the actor . These are the elementary business process use cases. For example, Process Sale is a concrete use case.

An **abstract use case** is never instantiated by itself; it is a subfunction use case that is part of another use case. Handle Credit Payment is abstract; it doesn't stand on its own, but is always part of another story, such as Process Sale.

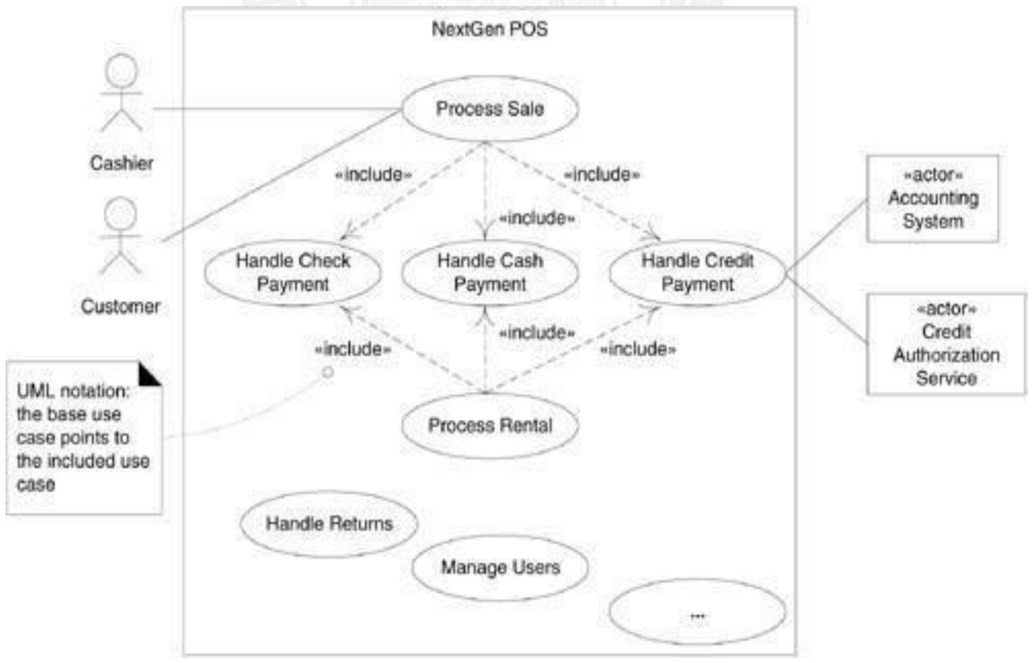
A use case that includes another use case, or that is extended or specialized by another use case is called **a base use case**. Process Sale is a base use case with respect to the included Handle Credit Payment subfunction use case. On the other hand, the use case that is an inclusion, extension, or specialization is called an addition use case. Handle Credit Payment is the addition use case in the include relationship to Process Sale. Addition use cases are usually abstract. Base use cases are usually concrete.

2. The extend Relationship

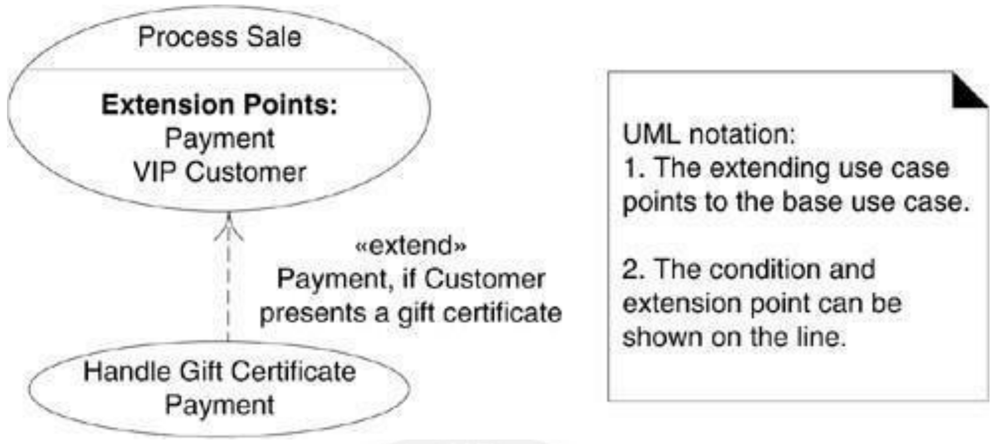
The idea is to create an extending or addition use case, and within it, describe where and under what condition it extends the behavior of some base use case.

The use of an extension point, and that the extending use case is triggered by some condition. Extension points are labels in the base use case which the extending use case references as the point of extension, so that the step numbering of the base use case can change without affecting the extending use case.

Use case include relationship in the Use-Case Model.



The extend relationship.


Process Sale
Extension Points:
Payment
VIP Customer
«extend»
Payment, if Customer presents a gift certificate
Handle Gift Certificate Payment
UML notation:
1. The extending use case points to the base use case.
2. The condition and extension point can be shown on the line.

3. <u>The Generalization Relationship</u>

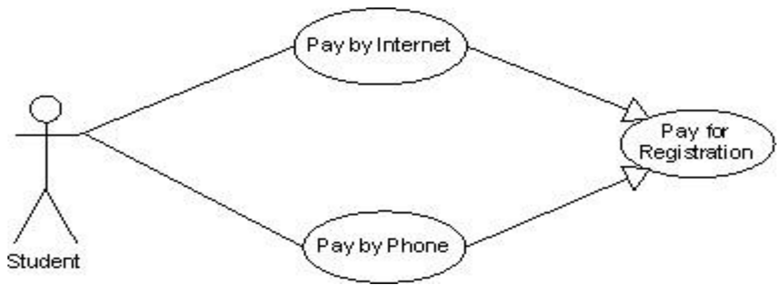
In the context of use case modeling the use case generalization refers to the relationship which can exist between two use cases and which shows that one use case (child) inherits the structure, behavior, and relationships of another actor (parent). The child use case is also referred to the more specialized use case while the parent is also referred to as the more abstract use case of the relationship.

For those of you familiar with object oriented concepts: use cases in UML are classes and the generalization is simply the inheritance relationship between two use cases by which one use case inherits all the properties and relationships of another use case.

You can use the generalization relationship when you find two or more use cases which have common behavior/logic. In this instance, you can describe the common parts in a separate use case (the parent) which then is specialized into two or more specialized child use cases.

Example:

If you are creating a payment system which allows students of a training provider to pay for courses both on-line and by phone, there will many things in common between the two scenarios: specifying personal info, specifying payment info, etc. However, there would also be differences between the two. So, the best way to accomplish this is to create one use case (the parent) which contains the common behavior and then create two specialized child use cases which inherit from the parent and which contain the differences specific to registering on-line vs. by phone.



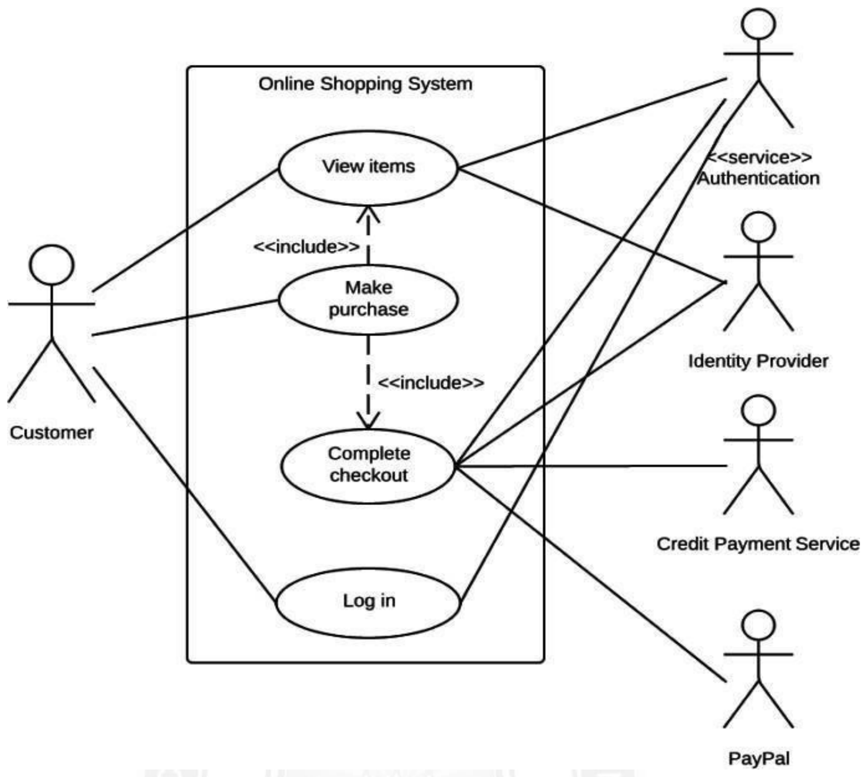
WHEN TO USE USECASE DIAGRAM

When to Use USECASE DIAGRAM

UML use case diagrams are ideal for:

- Representing the goals of system-user interactions
- Defining and organizing functional requirements in a system
- Specifying the context and requirements of a system
- Modeling the basic flow of events in a use case
- Reverse engineering.
- Forward engineering.

  Ex: Online shopping System



Use case diagrams are valuable for visualizing the functional requirements of a system that will translate into design choices and development priorities. They also help identify any internal or external factors that may influence the system and should be taken into consideration.

Use case diagrams specify the events of a system and their flows. But use case diagram never describes how they are implemented. Use case diagram can be imagined as a black box where only the input, output, and the function of the black box is known. Although use case is not a good candidate for forward and reverse engineering, still they are used in a slightly different way to make forward and reverse engineering. The same is true for reverse engineering. Use case diagram is used differently to make it suitable for reverse engineering. In forward engineering, use case diagrams are used to make test cases and in reverse engineering use cases are used to prepare the requirement details from the existing application.