**Use Case Relationship**

Use case relationship models the dependency between the use cases in the interaction model of a system. Well, the independent uses cases adequately represent the simpler system. But to represent the complex or large system we need to structure the complex use cases.

In this section, we will discuss in brief about modelling the relationship between use cases to represent a large and complex system. We will also discuss the kind of relationship that can be structured between the use cases.

**What is Use Case Relationship?**

In our previous content, we have discussed the[use case diagram](https://binaryterms.com/use-case-model.html) which represents the working of the system. A use case diagram has a set of ‘use cases’ and ‘actors’. A use case represents a bit of systems functionality and actors are the objects which interact with the system to get the work done from the system for which it is designed.

The independent use cases can represent a simple application but to represent the complex or large applications the system analyst needs to structure the complex relationship between the use cases.

Establishing relationships between the use cases allow **reusing** those use case which needs to be defined repeatedly and this reduces the efforts of developers.

The kinds of relations that could be modelled between use cases are as follow:

* Include relationship
* Extend relationship
* Generalization

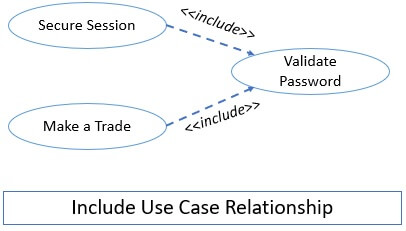
Include Relationship

Include relationship is modelled between the use cases when a use case **includes** the behavior sequence of another use case. The use cases that need to be described repeatedly for a complex or a large system are modelled once and included in the other uses cases when required.

Include use case is similar to **subroutines**, as they are the set of instructions which are repeatedly used in the program, but this set of instructions is written only once and are called whenever required. The include use case may or may not appear as behavior sequence on its own.

Let’s move on an example which would help us in understanding the include relationship among the use cases more clearly. Now, consider an online stock brokerage system which involves several uses cases among which three would be secure session, make a trade and validate password.

Now, whenever a user establishes a secure session it includes validation of passwords and whenever the user make a stock trade then also the passwords are validated.



So, the use cases secure session **includes** the use case ‘validate password’ and similarly make a trade use case also includes validate password use case. You can observe in the figure above that though use case validate password is repeatedly used by other use cases and it is modelled only once.

The include relationship is presented with a dashed arrow in UML diagram. The arrow head is pointing towards the included use case. Like, in the above figure the arrow head is pointed towards the validate password. The dashed arrow is always annotated by the keyword <<*include*>> to show the include relationship.

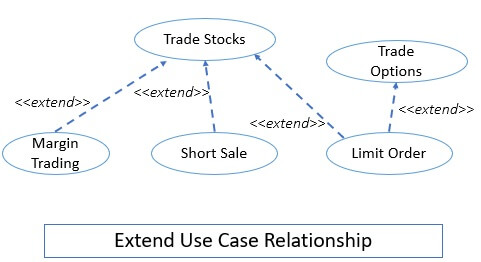
Extend Relationship

Extend relationship adds up behavior sequence to the base use case. It is somewhat similar to the include relationship but the direction of adding behaviour sequence to the base use case is opposite.

In the include relationship, the base use case incorporates the include use case. But in the extend relationship, the extended use case adds its behaviour sequence to the base use case.

Well, when we observe both ‘include’ and ‘extended’ relationship both add up behavior sequence to the base use case. This is the scenario when initially the basic elements are modelled and then advanced features are added to the base.

Let’s take an example of extend the relationship to understand it properly. A stock brokerage system has a use case trade stock, which allows the customer to buy the stock for the cash in the account. Now, if the customer has insufficient balance in his account but still, he wants to purchase the stock.



For this situation, the stock brokerage system adds a feature or extension, margin trade which allows the customer to take a loan to purchase the stock. This additional feature is added to the base use case **when** (condition) the stocks purchase cost is checked against the balance available in the customer account and it is found that the balance is insufficient to purchase the stock.

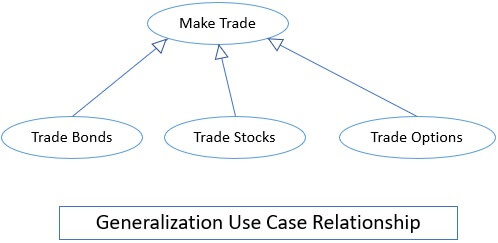
The extend relationship in UML notation is represented with the dashed arrow. The arrow head points to the base use case. This dashed arrow is annotated by the keyword <<*extend*>>.

The extension use case does not appear by its own rather the base use case must be valid even in absence of any extension use case. The extension use case appears at a specific point in the behavior sequence of the base class, most of the time it is a condition. When the condition becomes true the extension use case’s behavioral sequence occurs.

Generalization

Generalization is the most common term in computers. In generalization, there is a parent class consisting of common attributes and operations and the child class contains the refined attributes and operations.

Analogues to generalization in classes there is a concept of generalization in use case relationship. The parent use case contains the common behavior sequence and the child use case contains the refined features. Generalization in the use case is represented with the triangular arrow where the arrow head points to the parent use case.



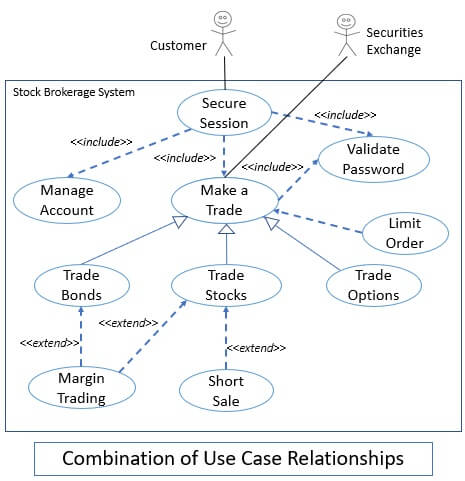
Well, the generalization in the use case is little complex when compared with generalization among classes. The child class add up attributes and operation to the parent class but the order of insertion is not so important.

While the child use case adds its behavior sequence in a specific location within the behavior sequence of the parent class.

In classes, we also have the concept of multiple inheritance but we can’t implement such a complex concept in use cases. Rather the include and extend relationship eliminates the need for multiple inheritance.

**Combination of Use Case Relationship**

A single-use case diagram can incorporate several use case relationships. The figure below shows the use case diagram of the stock brokerage system which shows how a use case diagram represents several relationships between the use cases.



Key Takeaways

* Use case relationship defines the complex relation between the use cases.
* Include use case is one where a use case incorporates another use case.
* In UML notation the include relationship annotated with the <<*include*>> keyword and a dashed arrow pointing towards the include use case.
* Include relationship may or may not appear on its own as a behavior sequence.
* Extend relationship is one where the extension use case adds itself to the base use case.
* In UML notation the extend relationship is annotated with the <<*extend*>> keyword and a dashed arrow pointing towards the base use case.
* The extend use case never appear on its own as a behavior sequence. It occurs at a specific point in the base use case behavior sequence, in most cases that point is a condition.
* Generalization in the use case is similar to generalization in classes. But, the order of adding the behavior sequence of a child to parents’ behavior sequence is important in the use case.
* A single use case diagram incorporates several use case relationships.