Use Case Diagram

Learn about the use case diagram and its benefits with some examples.

We'll cover the following

- Components of a use case diagram
- Relationships in use case diagrams
- Benefits of use case diagrams

Use case diagram describes the specification of users and their possible interactions with the system. These possible interactions are called **use cases.**

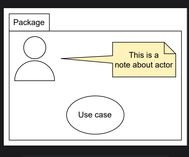
Components of a use case diagram

To understand a use case diagram, it is important to first understand its components. Use case diagrams have the following components:

- **Actor:** Users are called actors. They interact with the system. They could be humans, machines/hardware, or other external systems. There are two types of actors:
  - **Primary actors:** These are the humans or external systems that interact with that system and are responsible for initiating the use case. They are placed on the left side in a use case diagram. Primary actors are also called **active actors**.
  - **Secondary actors:** These are the ones that are used by the system to assist the primary actors in a use case. They cannot interact with the system on their own. They need primary actors to initiate a use case. Secondary actors are also called **passive actors**, and they are placed on the right side in a use case diagram.
- **Use case:** This is a single function performed on a system by an actor. It is represented by an oval shape.

- **Package:** This is a group of different elements. These groups are represented inside a folder icon.
- **Note:** This is used to add additional information about any component or relationship in a use case diagram.

The representation of the components explained above in a use case diagram is given below:



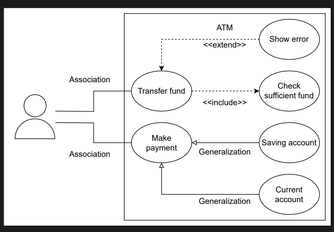
Components of a use case diagram

Relationships in use case diagrams

There are four different types of relationships in a use case diagram:

1. **Association:** This shows the relationship between and among actor(s) and use case(s). It represents how an actor can perform certain functions. It is denoted by a solid line without arrows. All the actors in a use case diagram must have at least one association with any use case. More than one actor can be associated with the same use case, and a single actor can be associated with more than one use case.
2. **Generalization:** This relationship is also known as inheritance. In inheritance, we have parent and children classes. Similarly, in a use case diagram, we have parent and child use cases. The child use case has generalization with the parent use case. Each child inherits the behavior of its parent. It is denoted by a solid line with an arrow on only one side (toward the parent use case).
3. **Include:** We use this to show the relationship between two use cases. It shows that one use case includes the behavior of another use case. The included use case will execute only after the execution of the base use case. We can also say that the base use case requires an included use case in order to be completed. It is represented by a dashed line with an arrow on only one side (toward the included use case), and we write `<<include>>` above the line.
4. **Extend:** We use this to show the relationship between two use cases. It shows that one use case extends the behaviors of another use case. The extended use case does not execute every time. It always depends on certain conditions. It is used to extend the functionality of the base use case. It is represented by a dashed line with an arrow on only one side (toward the base use case), and we write `<<extend>>` above the line.

In the example below, we have a small ATM (automated teller machine) transaction system where customers can transfer funds and make payments. To validate the funds, the transfer system has to check if a sufficient amount of funds is available. Otherwise, an error message will be displayed. To make a payment, a customer has two choices. It can either pay via a current account or a savings account.



The use case diagram of an ATM

Benefits of use case diagrams

A use case diagram is important in the following scenarios:

- It explains the flow and objective of all use cases.
- It helps in understanding the high-level functional requirements of the system.
- It defines a system's context and needs.

- It explains system behavior from a user perspective.
- It explains the scope of the system.

In the next lesson, we will discuss the class diagram with its components in detail.

Back

Types of UML Diagrams

Complete

Next

Class Diagram