Unified Modeling Language (UML)

Introduction

UML (Unified Modeling Language) is a visual language for describing and modeling software systems, processes, and other complex systems. UML diagrams are graphical representations of these systems and processes, using symbols and notation to depict their structure and behavior. Some common types of UML diagrams include class diagrams, activity diagrams, sequence diagrams, and state diagrams. They help analyst and developers communicate design ideas and to better understand the relationships and interactions between different elements of a system.

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

Diagram Components

- **Actors:** The users that interact with a system. An actor can be a person, an organization, or an outside system that interacts with your application or system. They must be external objects that produce or consume data.
- **System:** A specific sequence of actions and interactions between actors and the system. A system may also be referred to as a scenario.
- **Goals:** The end result of most use cases. A successful diagram should describe the activities and variants used to reach the goal.

Modeling Symbol Definitions

1. **Use cases:** Horizontally shaped ovals that represent the different actions a user might perform.
2. **Actors:** Stick figures that represent people or systems that are actually employing the use cases.
3. **Associations:** A line between actors and use cases. In complex diagrams, it is important to know which actors are associated with which use cases.
4. **System boundary boxes:** A box that sets a system scope to use cases. All use cases outside the box would be considered outside the scope of that system. For example, business central is outside the boundary of Assembly Picks.
5. **<<include>>:** A stereotype that is used to show a relationship between two use cases, where one use case includes the behavior of another. The symbol <<include>> is placed on an arrow that connects the including use case (the one that invokes the behavior) to the included use case (the one whose behavior is being invoked). This stereotype is used to capture the idea that the behavior of the included use case is part of the behavior of the including use case, and is used to simplify the overall structure of the model. The included use case is still a separate and distinct use case, but its behavior is incorporated into the behavior of the including use case as if it were part of that use case.
6. **<<extend>>:** A stereotype that is used to show a relationship between two use cases, where one use case extends the behavior of another. The symbol <<extend>> is placed on an arrow that connects the extending use case (the one that adds additional behavior) to the extended use case (the one whose behavior is being extended). This stereotype is used to capture the idea that the behavior of the extending use case is an optional addition to the behavior of the extended use case. The extending use case is still a separate and distinct use case, but its behavior is added to the behavior of the extended use case under specific conditions. For example, the

extending use case might only be applicable in certain situations, or it might only be available to certain users. In other words, the <<extend>> relationship represents an optional or conditional flow of behavior between two use cases.

Diagram Example