UNIT–II (12 Hrs.)

Requirements: Meta Model, Workflow, Functional and Non-functional Requirements; Requirement Attributes, Finding Requirements.

Use Case Modeling: Finding Actors and Use Cases,

Use Case Scenario – main flow, branching within a flow, repletion within a flow, modeling alternative flows; relationships among actors and use cases; use case diagrams

**Meta-Model Mechanism**

The UML is based on the four-level meta-modeling architecture. Each successive level is labeled from M3 to M0 and are usually named meta-metamodel, metamodel, class diagram, and object diagram respectively. A diagram at the Mi level is an instance of a diagram at the Mi+1 level. Therefore, an object diagram (an M0-level diagram) is an instance of some class diagram (an M1-level diagram), and this class diagram is an instance of a metamodel (an M2-level diagram). The M3-level diagram is used to define the structure of a metamodel, and the Meta Object Facility (MOF) belongs to this level. The UML metamodel belongs to the M2-level.

A screenshot of a cell phone

Description automatically generated

We define *model instantiation checking* to be the process in which an Mi-level diagram is checked to see if it is a correct instance of the corresponding Mi+1-level diagram that can claim it is an instance. A tool that can check a user-defined model against a UML metamodel and also an object diagram against a user-defined class diagram are developed. Label the Mi-level diagram as the *instance diagram* and the Mi+1-level diagram as the *specification diagram*.

A tool that translates a UML model (consisting of the specification diagram, OCL constraints, and the instance diagram) into an executable specification in order to perform model instantiation checking has been developed. The tool checks a user-defined model against a UML metamodel by converting the user-defined model into an object diagram based on the metamodel and checking to see if it is a valid object diagram with respect to the metamodel. The tool also checks an object diagram against a user-defined model. Model instantiation checking is done by checking graphical and OCL constraints provided in the specification class diagram to ensure that the instance diagram is valid.

The tool converts UML models into [Abstract State Machines](http://www.eecs.umich.edu/gasm/) (ASMs), An *Abstract State Machine* is a state machine that computes a set of updates of its own variables by firing all possible updates based on the current state. The computation of a set of updates occurs at the same time and results in the generation of a new state. ASMs can be formally defined and can be used to define precise models of software.

The high-level language [AsmL](http://research.microsoft.com/foundations/asml/" \t "_blank), which is an executable specification language based on the concept of abstract state machines that is developed by the [Foundations of Software Engineering (FSE)](http://research.microsoft.com/fse/) group at Microsoft Research is used. It is a high-level specification language running on Microsoft's .NET framework and has language constructs such as sets and sequences and high-level operations that let the programmer specify what the program should do but not how it should be done. AsmL is also object-oriented so it is easy to translate UML classes and their features into AsmL.

**What is a Functional Requirement?**

In software engineering, a functional requirement defines a system or its component. It describes the functions a software must perform. A function is nothing but inputs, its behavior, and outputs. It can be a calculation, data manipulation, business process, user interaction, or any other specific functionality which defines what function a system is likely to perform.

Functional software requirements help to capture the intended behavior of the system. This behavior may be expressed as functions, services or tasks or which system is required to perform.

**Example of Functional Requirements**

* The software automatically validates customers against the ABC Contact Management System
* The Sales system should allow users to record customers sales
* The background color for all windows in the application will be blue and have a hexadecimal RGB color value of 0x0000FF.
* Only Managerial level employees have the right to view revenue data.
* The software system should be integrated with banking API
* The software system should pass [Section 508](https://www.section508.gov/) accessibility requirement.

What is Non-Functional Requirement?

A non-functional requirement defines the quality attribute of a software system. They represent a set of standards used to judge the specific operation of a system. Example, how fast does the website load?

A non-functional requirement is essential to ensure the usability and effectiveness of the entire software system. Failing to meet non-functional requirements can result in systems that fail to satisfy user needs.

Non-functional Requirements allows you to impose constraints or restrictions on the design of the system across the various agile backlogs. Example, the site should load in 3 seconds when the number of simultaneous users are > 10000.

**Examples of Non-functional requirements**

Here, are some examples of non-functional requirement:

1. Users must change the initially assigned login password immediately after the first successful login. Moreover, the initial should never be reused.
2. Employees never allowed to update their salary information. Such attempt should be reported to the security administrator.
3. Every unsuccessful attempt by a user to access an item of data shall be recorded on an audit trail.
4. A website should be capable enough to handle 20 million users with affecting its performance
5. The software should be portable. So moving from one OS to other OS does not create any problem.
6. Privacy of information, the export of restricted technologies, intellectual property rights, etc. should be audited.

Introduction

Use case modeling (use case diagrams) describes what a system does or what is the functionality provided by the system to benefit the users. Use case modeling was created by Ivar Jacobson.

More than any other diagrams in UML, use case diagrams allow us to quickly gather the requirements of the software system. The primary components of a use case model are use cases, actors or roles and the system being modeled also known as the subject.

The primary purpose of use cases are:

1. To describe the functional requirements of the system, resulting in an agreement between the stakeholders and the software developers who are developing the system.
2. To give a clear and consistent description of what the system should do.
3. To provide a basis for conducting system tests to verify whether the system works appropriately or not.
4. To provide the ability to transform functional requirements into classes and operations in the system.

Use Case

A use case represents the functionality provided by the system to the user. A use case is defined as “a set of actions performed by the system, which produces an observable result that is, typically, of some value to one or more actors or other stakeholders of the system”.

The actions can include communicating with other actors or systems as well as performing calculations inside the system. The characteristics of a use case are:

1. A use case is always initiated by an actor.
2. A use case provides value to an actor.
3. A use case is complete.

Use cases are connected to actors through associations, which are sometimes referred to as communication associations. Associations represent which actors the use case is communicating with. The association should always be binary, implying a dialog between the actor and system.

A use case is a classifier, not an instance. A use case represents the functionality as a whole with possible alternatives, errors and exceptions that can occur during the execution of the use case. An instance of the use case is known as a scenario, which represents a specific execution path through the system.

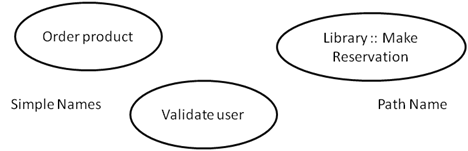
Example: Consider a online shopping system

Use case: Purchase Product

Scenario: A user named surya purchases a product, gaming console by using debit card payment option.

The graphical representation of a use case is a solid ellipse with the use case name specified inside the ellipse. A use case generally placed inside the system boundary and is always connected to an actor using an association relationship.

The name of the use case is generally a phrase rather a one word label. The name can be a simple name or a path name as shown in the below example:

[](http://www.startertutorials.com/uml/wp-content/uploads/2013/08/1-uml-usecases.gif)

Actors

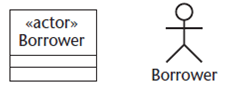
An actor is one which interacts or uses the system (subject). An actor sends or receives messages from the system. Actor can be either a person or another system (computer or application).

An actor is a classifier, not an instance. An actor represents a role, not an individual user of the system. For example, in an online shopping system, if ramesh wants to purchase a product, his role will be buyer.

An actor has a name, and the name should reflect the actor’s role. The name should not represent an instance of the actor or the functionality of an actor. A use case is always initiated by an actor that sends a message to it. This message is also known as stimulus.

Actors may be of two types: active actors and passive actors. Active actors or those which initiates a use case and passive actors are those which participate in a use case but never initiate it.

Actor can be represented with a class symbol stereotyped as <<actor>>. Actor has a standard stereotype icon known as the “stickman”. An actor can have both attributes and behavior.

[](http://www.startertutorials.com/uml/wp-content/uploads/2013/08/2-actor-use-case.gif)

**Flow of Events**

A use case describes what a system does but it does not specify how it does that. We can specify the behavior of a use case by describing the flow of events in text clearly enough for an outsider to understand it easily.

When we write the flow of events, we must specify when the use case starts and ends, what objects are exchanged between the system and the actor, the basic/main flow and alternate flows of events. For example in an ATM system, we can describe the use case “Validate User” in the following way:

***Main flow of events:***

The use case starts when the system prompts the customer for a PIN number. The customer can now enter a PIN number via the keypad. The customer commits the entry by pressing the Enter button. The system then checks this PIN number to see if its valid or not. If the PIN is valid, the system acknowledges the entry, thus ending the use case.

***Exceptional flow of events:***

Customer can cancel the transaction at any point by pressing the Cancel button thereby restarting the use case. No changes are made to the customer’s state.

***Exceptional flow of events:***

Customer can clear the PIN number anytime before confirming it and reenter a new PIN number again.

***Exceptional flow of events:***

If the Customer enters an invalid PIN number, the use case restarts. If this happens, three times in a row, the system cancels the transaction, preventing the Customer from interacting with the ATM for 60 seconds.

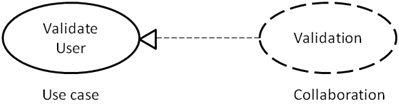
**Scenarios**

A scenario is a specific sequence of actions that illustrates behavior. Scenario is an instance of an use case like objects are instances of classes. For example, in an online shopping system consider the use case “Purchase Product”.

For this use case, a scenario can be: user Ramesh purchases a laptop. Another scenario can be user Mahesh purchases a washing machine etc..

**Collaborations**

A use case specifies what the system does but does not specify how it is implemented. A use case is implemented by creating a collection of classes and other elements that work together to achieve the behavior of the use case. This collection of elements, including both its static and dynamic structure, is modeled in UML as collaboration.

[](http://www.startertutorials.com/uml/wp-content/uploads/2013/08/3-use-case-collaboration.gif)

**Organizing Use Cases**

We can organize the use cases by grouping them in packages in the same manner in which we can organize classes. We can also organize use cases by specifying generalization, include and extend relationships. We apply these relationships in order to factor common behavior and in order to factor variants (alternatives).

Generalization among use case represents that the child use case inherits the behavior from the parent use case. For example in an ATM system, the behavior of the use case “Validate User” is to check whether the user is a valid user or not.

To implement this, system might ask for a PIN number or might ask for a retinal scan of the eye or may ask for finger scan. All these three, PIN validation, Retinal scan and Finger scan are specialized ways of checking the validity of a user and can be applied at any place where the “Validate User” use case appears.

An include relationship between use cases means that the base use case explicitly incorporates the behavior of the included use case. Such relationship is represented as a dependency stereotyped with include. In an include relationship, the included use case cannot exist without the base use case. The include relationship is used to separate the common behavior.

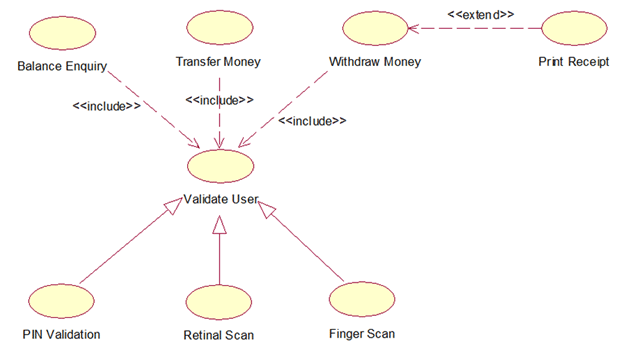
[](http://www.startertutorials.com/uml/wp-content/uploads/2013/08/4-use-case-incude.gif)

An extend relationship between use cases means that the extending use case extends the behavior of the base use case. Such relationship is graphically represented as dependency stereotyped with exclude.

In a extend relationship, the base use case may exist on its own but at certain points its behavior may be extended by the behavior of another use case. The extend relationship is used to separate the optional behavior.

[](http://www.startertutorials.com/uml/wp-content/uploads/2013/08/5-use-case-extend.gif)

Consider the following which illustrates the use of generalization, include and extend relationships between use cases:

[](http://www.startertutorials.com/uml/wp-content/uploads/2013/08/6-organizing-use-cases.gif)

Common Modeling Techniques

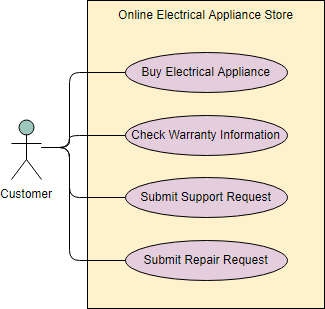
Modeling the behavior of an element

To model the behavior of an element:

1. Identify the actors that interact with the element.
2. Organize the actors by identifying the general and more specialized roles.
3. For each actor, consider the primary ways in which the actor interacts with the element.
4. Consider also the alternative ways in which the actor interacts with the element.
5. Organize these behaviors as use cases, applying include and extend relationships to factor common behavior and distinguish exceptional behavior.

# Use Case Diagram Tutorial

A use case describes how a user uses a system to accomplish a particular goal. A use case diagram consists of the system, the related use cases and actors and relates these to each other to visualize: what is being described? (**system**), who is using the system? (**actors**) and what do the actors want to achieve? (**use cases**), thus, use cases help ensure that the correct system is developed by capturing the requirements from the user's point of view.



## What is a Use Case Diagram in UML?

A use case is a list of actions or event steps typically defining the interactions between a role of an actor and a system to achieve a goal. A use case is a useful technique for identifying, clarifying, and organizing system requirements. A use case is made up of a set of possible sequences of interactions between systems and users that defines the features to be implemented and the resolution of any errors that may be encountered.

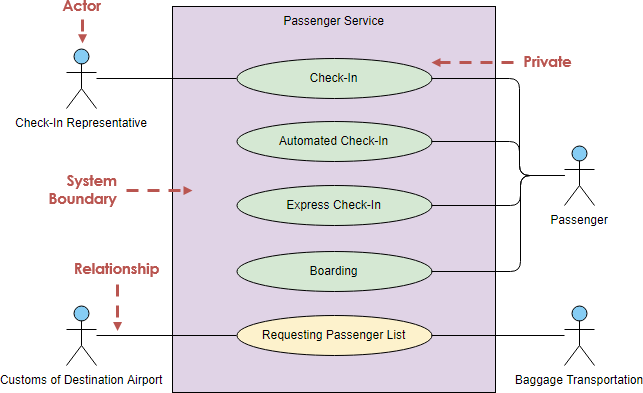
While a use case itself might drill into a lot of detail (such as, flow of events and scenarios) about every possibility, a use-case diagram can help provide a higher-level view of the system, providing the simplified and graphical representation of what the system must actually do.

A use case (or set of use cases) has these characteristics:

* Organizes functional requirements
* Models the goals of system/actor (user) interactions
* Describes one main flow of events (main scenarios) and possibly other exceptional flows (alternatives), also called paths or user scenarios

## Use Case Diagram Notations

Use cases define interactions between external actors and the system to attain particular goals. A use case diagram contains four main components



### Actor

Actors are usually individuals involved with the system defined according to their roles. The actor can be a human or other external system.

### Use Case

A use case describes how actors uses a system to accomplish a particular goal. Use cases are typically initiated by a user to fulfill goals describing the activities and variants involved in attaining the goal.

### Relationship

The relationships between and among the actors and the use cases.

### System Boundary

The system boundary defines the system of interest in relation to the world around it.

## Benefits of Use Case Diagram

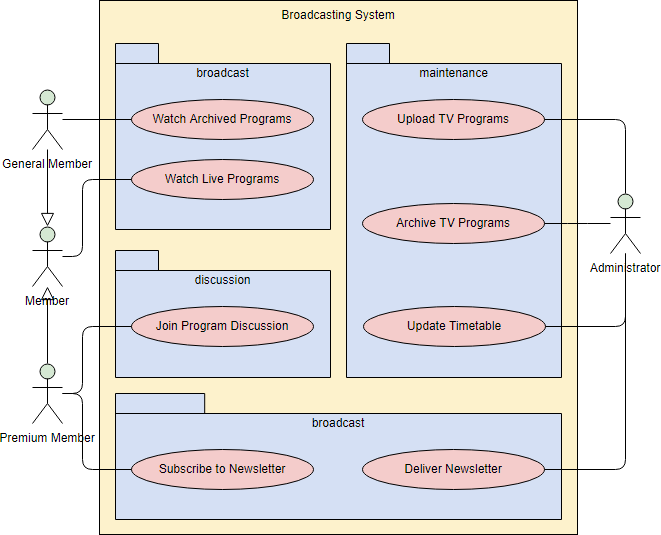
* Use cases is a powerful technique for the elicitation and documentation of black-box functional requirements.
* Because, use cases are easy to understand and provide an excellent way for communicating with customers and users as they are written in natural language.
* Use cases can help manage the complexity of large projects by partitioning the problem into major user features (i.e., use cases) and by specifying applications from the users' perspective.
* A use case scenario, often represented by a sequence diagram, involves the collaboration of multiple objects and classes, use cases help identify the messages (operations and the information or data required - parameters) that glue the objects and classes together.
* Use cases provide a good basis to link between the verification of the higher-level models (i.e. interaction between actors and a set of collaborative objects), and subsequently, for the validation of the functional requirements (i.e. blueprint of white-box test).
* Use case driven approach provides an traceable links for project tracking in which the key development activities such as the use cases implemented, tested, and delivered fulfilling the goals and objectives from the user point of views.

**How to Draw a Use Case Diagram?**

A Use Case model can be developed by following the steps below.

1. Identify the Actors (role of users) of the system.
2. For each category of users, identify all roles played by the users relevant to the system.
3. Identify what are the users required the system to be performed to achieve these goals.
4. Create use cases for every goal.
5. Structure the use cases.
6. Prioritize, review, estimate and validate the users.

Note that: to make use case approach more "Agile", do not detail all use cases, but prioritize them in your product backlog, you should refine the use case in different level of details according to the development phase with just-in-time and just-enough manner.

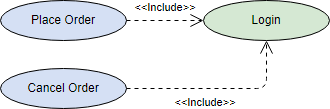
* Draw packages for logical categorization of use cases into related subsystems.  
  

## Structuring Use Cases

UML defines three stereotypes of association between Use Cases:

### <<include>> Use Case

The time to use the <<include>> relationship is after you have completed the first cut description of all your main Use Cases. You can now look at the Use Cases and identify common sequences of user-system interaction.



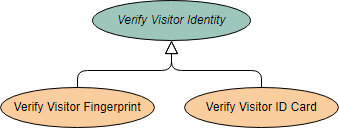
### <<extend>> Use Case

An extending use case is, effectively, an alternate course of the base use case. The <<extend>> use case accomplishes this by conceptually inserting additional action sequences into the base use-case sequence.

UML Use Case Diagram Extend Use Case Example

### Abstract and generalized Use Case

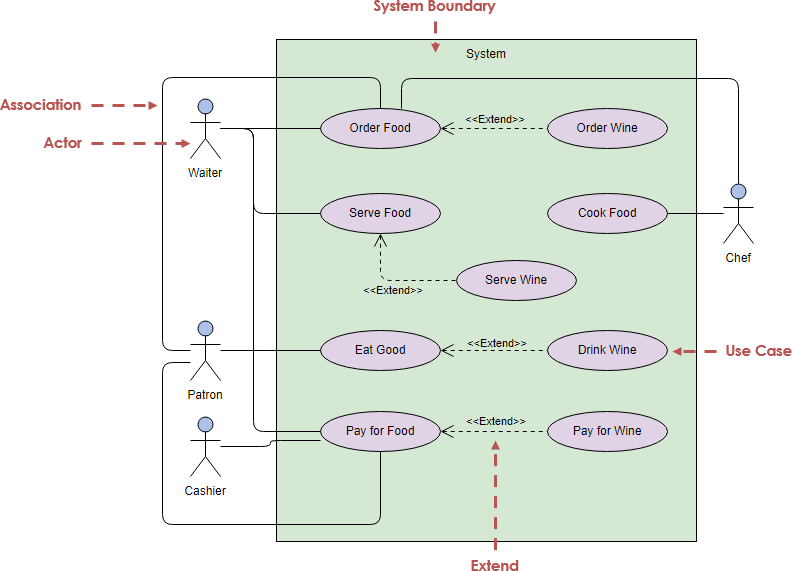
The general use case is abstract. It can not be instantiated, as it contains incomplete information. The title of an abstract use case is shown in italics.



**Example**

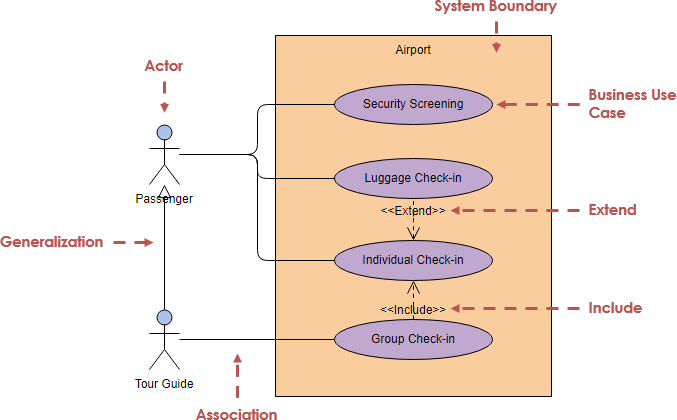
This example depicts a model of several business use cases (goals) which represents the interactions between a restaurant (the business system) and its primary actors.

After the base use cases have been identified in the first cut, perhaps we could further structuring those use case with <<extend>> and <<include>> use cases in the second round touch up as shown in the Figure below:



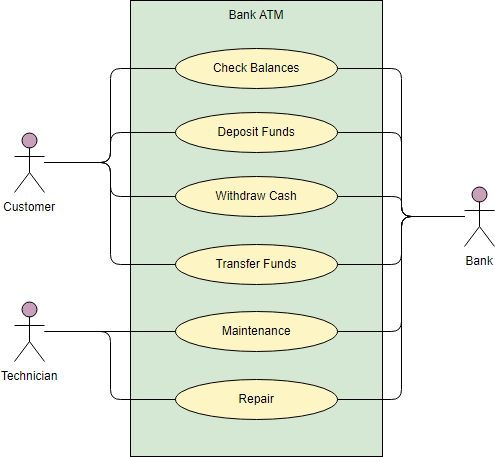
## Business Use Case

A business use case is described in **technology-free terminology** which treats the business process as a black box and describes the business process that is used by its business actors, while an ordinary use case is normally described at the **system functionality level** and specifies the function or the service that the system provides for the user. In other words, business use case represents how the work to be done manually in the currently situation and it is not necessarily done by the system or intend to be automated in the scope of target system.

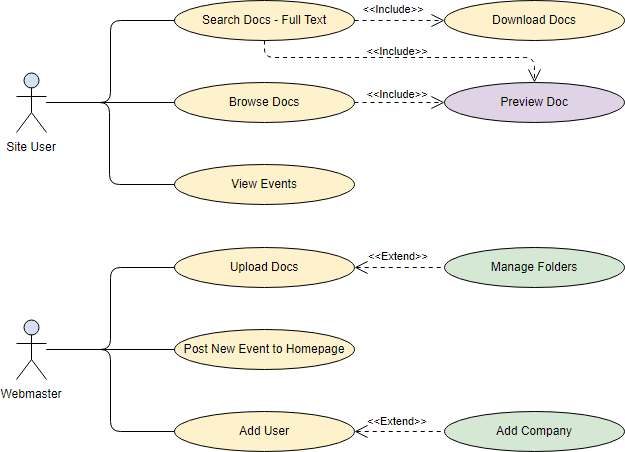


## Use Case Diagram Examples

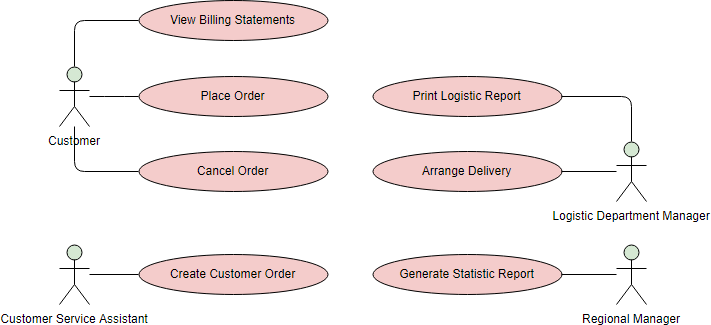
The figure below shows an **ATM** use case diagram example, which is quite a classic example to use in teaching use case diagram.

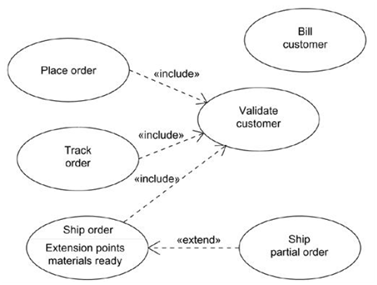


The **Document Management System (DMS)** use case diagram example below shows the actors and use cases of the system. In particular, there are include and extend relationships among use cases.



The **Order System** use case diagram example below shows the actors and use cases involved in the system:



[](http://www.startertutorials.com/uml/wp-content/uploads/2013/08/7-modeling-behavior-of-an-element.gif)