Modeling as a Design Technique

A model is an abstraction of something for the purpose of understanding it before building it. Because a model omits nonessential details, it is easier to manipulate than the original entity. Abstraction is fundamental human capability that permits us to deal with complexity. Engineers, artists, and craftsmen have built models for thousands of years to try out designs before executing them. Development of hardware and software systems is no exception. To build complex systems, the developer must abstract different views of the system. Build models using precise notations, verify that the models satisfy the requirements of the system, and gradually add details to transform the model into an implementation

Three type Models

We model a system from three related but different viewpoints, each capturing important aspects of the system, but all required for a complete description. The class model represents the static, structural, “data” aspects of a system. The state model represents the temporal, behavioral, “control” aspects of a system. The interaction model represents the collaboration of individual objects, the “interaction” aspects of a system.

A typical software procedure incorporates all three aspects:   
It uses data structures( Class Model), it sequences operations in time (state model), and it passes data and control among objects(interaction model). Each model contain references to entities in other models.

For example, class model attaches operations to classes, while the state and interaction models elaborate the operations.

The three kinds of models separate a system into distant views. The different models are not completely independent- A system is more than a collection of independent parts-but each model can be examined and understood by itself to a large extent.

Interaction Model

Interaction can be modeled at different levels of abstraction.

1. Use case Diagram
2. Sequence diagram
3. Activity diagram

Use Case Model

What is a Use Case?

A formal way of representing how a business system interacts with its environment .Illustrates the activities that are performed by the users of the system. A scenario-based technique in the UML. A sequence of actions a system performs that yields a valuable result for a particular actor.

What is an Actor?

A user or outside system that interacts with the system being designed in order to obtain some value from that interaction.

Use Cases describe scenarios that describe the interaction between users of the system (the actor) and the system itself.

Use case diagrams describe what a system does from the standpoint of an external observer. The emphasis is on what a system does rather than how. Use case diagrams are closely connected to scenarios. A scenario is an example of what happens when someone interacts with the system.

How to Draw Use Case Diagram?

Step 1 Identify the actors

As we read the scenario, define those people or systems that are going to interact with the scenario.

An Actor is outside or external the system.

It can be a:

• Human

• Peripheral device (hardware)

• External system or subsystem

• Time or time-based event

Represented by stick figure

Step 2 Identify the use case

Each use case in a use case diagram describes one and only one function in which users interact with the system

May contain several “paths” that a user can take while interacting with the system • Each path is referred to as a scenario

Labelled using a descriptive verb-noun phrase

Represented by an oval

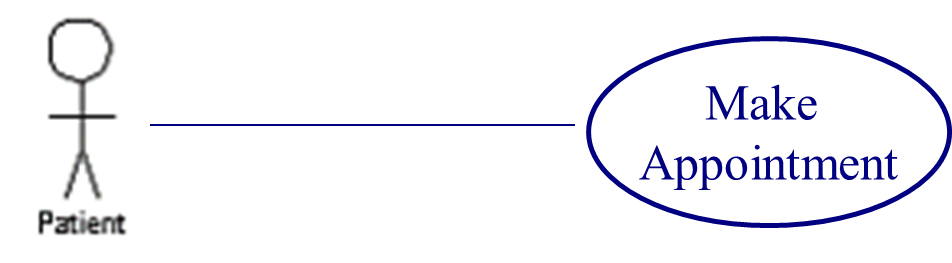
Use Case - Relationships

Relationships

Represent communication between actor and use case

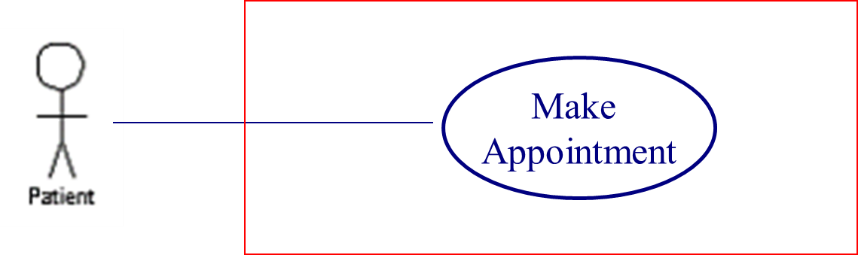
Depicted by line or double-headed arrow line

Also called association relationship



Boundary

A boundary rectangle is placed around the perimeter of the system to show how the actors communicate with the system.



* Boundary
  + A boundary rectangle is placed around the perimeter of the system to show how the actors communicate with the system.

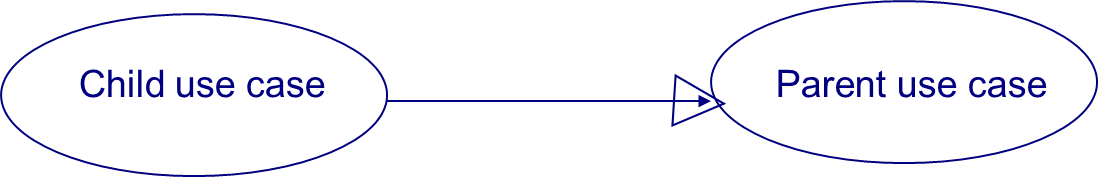
**Other Types of Relationships for Use Cases**

* 1. Generalization
  2. Include
  3. Extend

**Generalization Relationship**

Represented by a line and a hollow arrow

From child to parent



**Include Relationship**

Represents the inclusion of the functionality of one use case within another

Arrow is drawn from the base use case to the used use case

Write << include >> above arrowhead line

**Extend relationship**

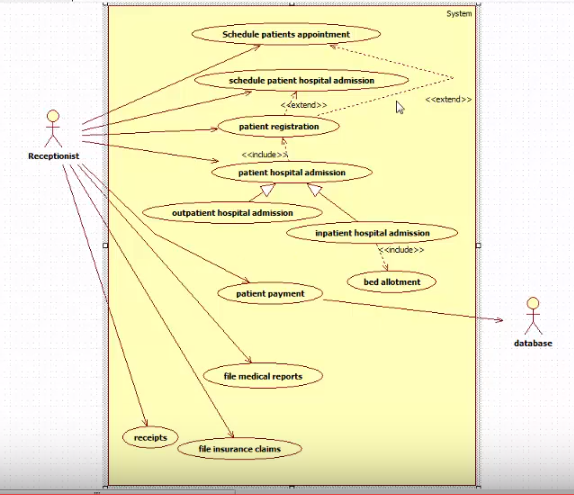
Represents the extension of the use case to include optional functionality

Arrow is drawn from the extension use case to the base use case

Write << extend >> above arrowhead line

**Practical No. 1**





**Practical No. 2**

****

