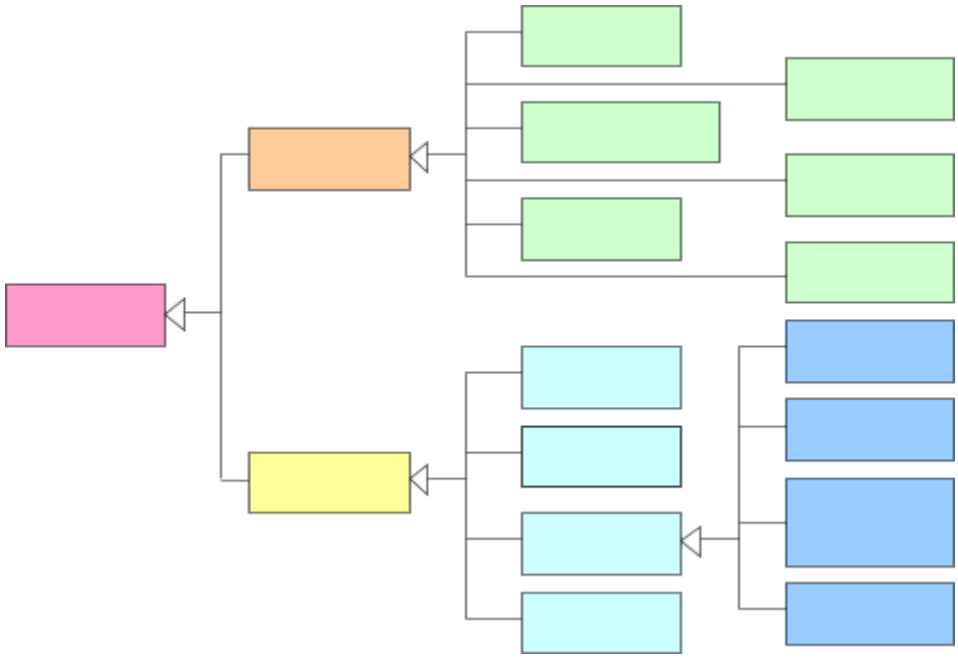
Types of UML Diagrams:



There are two broad categories of UML diagrams:

- Structural Diagrams
- Behavioral Diagrams

Structural Diagrams:

The structural diagrams represent the static aspect of the system. These static aspects represent those parts of a diagram which forms the main structure and therefore stable.

These static parts are represents by classes, interfaces, objects, components and nodes. The four structural diagrams are:

1. Class diagram
2. Object diagram
3. Component diagram
4. Deployment diagram

Behavioral Diagrams:

Behavioral diagrams basically capture the dynamic aspect of a system. Dynamic behaviour means the behaviour of the system when it is running /operating.

UML has the following five types of behavioral diagrams:

1. Use case diagram
2. Sequence diagram

3. Collaboration diagram
4. Statechart diagram
5. Activity diagram

Use Case Diagram:

The purposes of use case diagrams can be as follows:

- ❑ Used to gather requirements of a system.
- ❑ Used to get an outside view of a system.
- ❑ Identify external and internal factors influencing the system.
- ❑ Show the interacting among the requirements and actors.

Elements of a Use Case Diagram

A use case diagram is quite simple in nature and depicts two types of elements: one representing the business roles and the other representing the business processes. Let us take a closer look at use at what elements constitute a use case diagram.

Actors: An actor portrays any entity (or entities) that perform certain roles in a given system. The different roles the actor represents are the actual business roles of users in a given system. An actor in a use case diagram interacts with a use case. For example, for modeling a banking application, a customer entity represents an actor in the application. Similarly, the person who provides service at the counter is also an actor. But it is up to you to consider what actors make an impact on the functionality that you want to model. If an entity does not affect a certain piece of functionality that you are modeling, it makes no sense to represent it as an actor. An actor is shown as a stick figure in a use case diagram depicted "outside" the system boundary, as shown in Figure 3.1.

Figure 3.1: an actor in a use case diagram

To identify an actor, search in the problem statement for business terms that portray roles in the system. For example, in the statement "patients visit the doctor in the clinic for medical tests," "doctor" and "patients" are the business roles and can be easily identified as actors in the system.

Use case: A use case in a use case diagram is a visual representation of distinct business functionality in a system. The key term here is "distinct business functionality." To choose a business process as a likely candidate for modeling as a use case, you need to ensure that the business process is discrete in nature. As the first step in identifying use cases, you should list the

discrete business functions in your problem statement. Each of these business functions can be classified as a potential use case. Remember that identifying use cases is a discovery rather than a creation. As business functionality becomes clearer, the underlying use cases become more easily evident. A use case is shown as an ellipse in a use case diagram (see Figure 3.2).

Figure 3.2: use cases in a use case diagram

Figure 3.2 shows two uses cases: "Make appointment" and "Perform medical tests" in the use case diagram of a clinic system. As another example, consider that a business process such as "manage patient records" can in turn have sub-processes like "manage patient's personal information" and "manage patient's medical information." Discovering such implicit use cases is possible only with a thorough understanding of all the business processes of the system through discussions with potential users of the system and relevant domain knowledge.

System boundary: A system boundary defines the scope of what a system will be. A system cannot have infinite functionality. So, it follows that use cases also need to have definitive limits defined. A system boundary of a use case diagram defines the limits of the system. The system boundary is shown as a rectangle spanning all the use cases in the system.

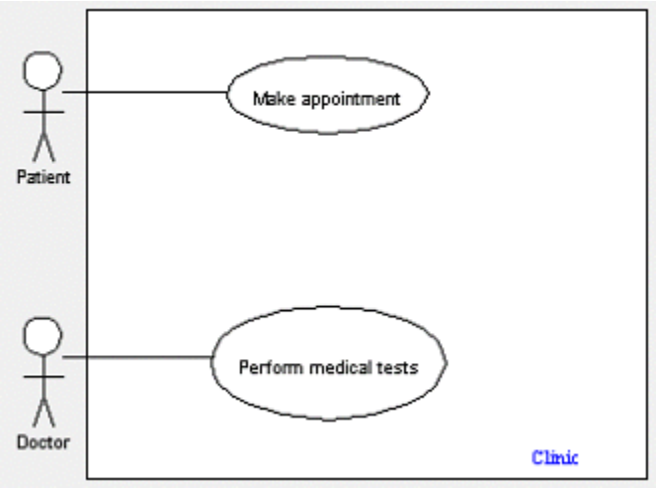


Figure 3.3: a use case diagram depicting the system boundary of a clinic application

Figure 3.3 shows the system boundary of the clinic application. The use cases of this system are enclosed in a rectangle. Note that the actors in the system are outside the system boundary.

The system boundary is potentially the entire system as defined in the problem statement. But this is not always the case. For large and complex systems, each of the modules may be the system boundary. For example, for an ERP system for an organization, each of the modules such as personnel, payroll, accounting, and so forth, can form the system boundary for use cases specific to each of these business functions. The entire system can span all of these modules depicting the overall system boundary.

How to draw Use Case Diagram?

- The name of a use case is very important. So the name should be chosen in such a way so that it can identify the functionalities performed.
- Give a suitable name for actors.
- Show relationships and dependencies clearly in the diagram.
- Do not try to include all types of relationships. Because the main purpose of the diagram is to identify requirements.
- Use note when ever required to clarify some important points.

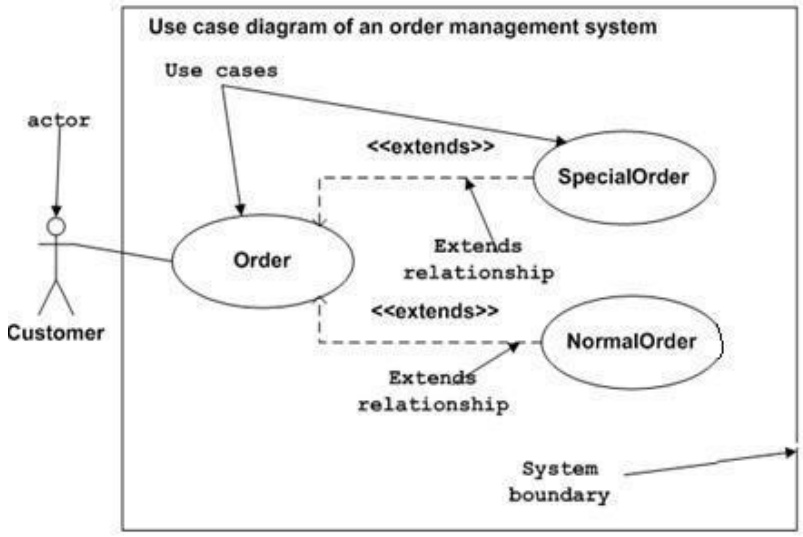


Figure: Sample Use Case diagram

Where to Use Case Diagrams?

Use case diagrams specify the events of a system and their flows. But use case diagram never describes how they are implemented. Use case diagram can be imagined as a black box where only the input, output and the function of the black box is known.

So the following are the places where use case diagrams are used:

- Requirement analysis and high level design.
- Model the context of a system.
- Reverse engineering.

- Forward engineering.

Relationships in Use Cases

Use cases share different kinds of relationships. A relationship between two use cases is basically a dependency between the two use cases. Defining a relationship between two use cases is the decision of the modeler of the use case diagram. This reuse of an existing use case using different types of relationships reduces the overall effort required in defining use cases in a system. A similar reuse established using relationships, will be apparent in the other UML diagrams as well. Use case relationships can be one of the following:

Include: When a use case is depicted as using the functionality of another use case in a diagram, this relationship between the use cases is named as an include relationship. Literally speaking, in an include relationship, a use case includes the functionality described in the another use case as a part of its business process flow. An include relationship is depicted with a directed arrow having a dotted shaft. The tip of the arrowhead points to the parent use case and the child use case is connected at the base of the arrow. The stereotype "<<include>>" identifies the relationship as an include relationship.

Figure 3.4: an example of an include relationship

For example, in Figure 3.4, you can see that the functionality defined by the "Validate patient records" use case is contained within the "Make appointment" use case. Hence, whenever the "Make appointment" use case executes, the business steps defined in the "Validate patient records" use case are also executed.

Extend: In an extend relationship between two use cases, the child use case adds to the existing functionality and characteristics of the parent use case. An extend relationship is depicted with a directed arrow having a dotted shaft, similar to the include relationship. The tip of the arrowhead points to the parent use case and the child use case is connected at the base of the arrow. The stereotype "<<extend>>" identifies the relationship as an extend relationship, as shown in Figure 3.5.

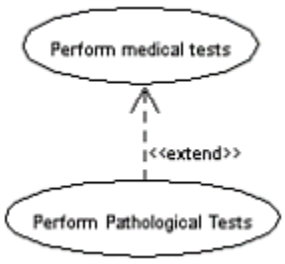


Figure 3.5: an example of an extend relation

Figure 3.5 shows an example of an extend relationship between the "Perform medical tests" (parent) and "Perform Pathological Tests" (child) use cases. The "Perform Pathological Tests" use case enhances the functionality of the "Perform medical tests" use case. Essentially, the "Perform Pathological Tests" use case is a specialized version of the generic "Perform medical tests" use case.

Generalizations: A generalization relationship is also a parent-child relationship between use cases. The child use case in the generalization relationship has the underlying business process meaning, but is an enhancement of the parent use case. In a use case diagram, generalization is shown as a directed arrow with a triangle arrowhead (see Figure 3.6). The child use case is connected at the base of the arrow. The tip of the arrow is connected to the parent use case.

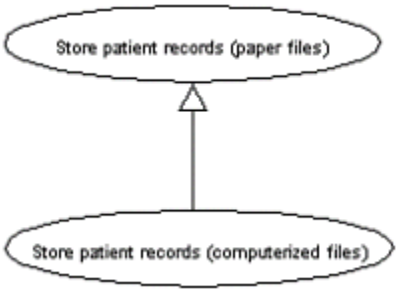


Figure 3.6: an example of a generalization relationship

On the face of it, both generalizations and extends appear to be more or less similar. But there is a subtle difference between a generalization relationship and an extend relationship. When you establish a generalization relationship between use cases, this implies that the parent use case can be replaced by the child use case without breaking the business flow. On the other hand, an extend relationship between use cases implies that the child use case enhances the functionality of the parent use case into a specialized functionality. The parent use case in an extend relationship cannot be replaced by the child use case.

Let us see if we understand things better with an example. From the diagram of a generalization relationship (refer to Figure 3.6), you can see that "Store patient records (paper file)" (parent) use case is depicted as a generalized version of the "Store patient records (computerized file)" (child) use case. Defining a generalization relationship between the two implies that you can replace any occurrence of the "Store patient records (paper file)" use case in the business flow of your system with the "Store patient records (computerized file)" use case without impacting any business flow. This would mean that in future you might choose to store patient records in a computerized file instead of as paper documents without impacting other business actions.

Now, if we had defined this as an extend relationship between the two use cases, this would imply that the "Store patient records (computerized file)" use case is a specialized version of the "Store patient records (paper file)" use case. Hence, you would not be able to seamlessly replace the occurrence of the "Store patient records (paper file)" use case with the "Store patient records (computerized file)" use case.

Writing a Use Case Specification

Use case ID: UC1.1
Use case Name: User Login

Actor: User

Description:

1. Input user ID and password.
2. Check User ID and password (User authentication).
3. Save ID and Password (Cookies).
4. Remember Password.

Exception:

1. Page not found.
2. No database connection.

Precondition:

1. URL of login page from web browser.

Post Condition:

1. **Successful login –** Logging in to system**.**
2. **Unsuccessful login –** Stay in same page.
3. **Exception -** Stay in same page if possible.