
Object-Oriented Analysis and Design using JAVA

B.Tech (CSE/IT) 5th SEM
2021-2022

Lecture-12 Requirement elicitation and use cases

Introduction

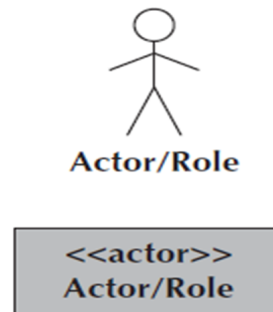
There are many activities performed during requirements elicitation. These activities map a problem statement into a requirements specification which is represented as a set of actors, scenarios, and use cases. Requirements elicitation activities include:

- Identifying Actors
- Identifying Scenarios
- Identifying Use Cases
- Refining Use Cases
- Identifying Relationships Among Actors and Use Cases
- Identifying Initial Analysis Objects
- Identifying Non functional Requirements

Identifying actors

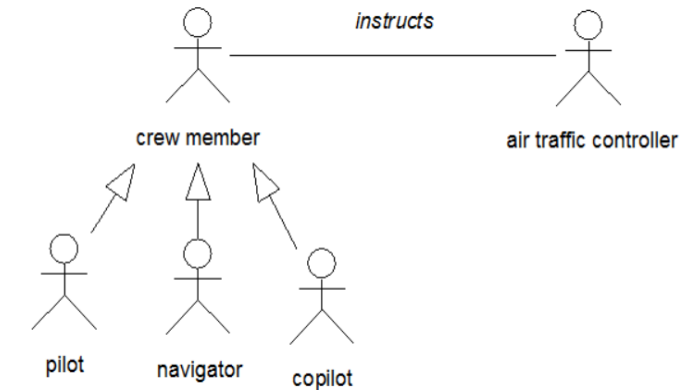
Actors represent external entities that interact with the system. An actor can be human or an external system or actor is a person or system that derives benefit from and is external to the subject.

- Is depicted as either a stick figure (default) or, if a nonhuman actor is involved, a rectangle with <<actor>> in it (alternative).
- Is labeled with its role.
- Can be associated with other actors using a specialization/superclass association, denoted by an arrow with a hollow arrowhead.
- Is placed outside the subject boundary.



Questions for identifying actors

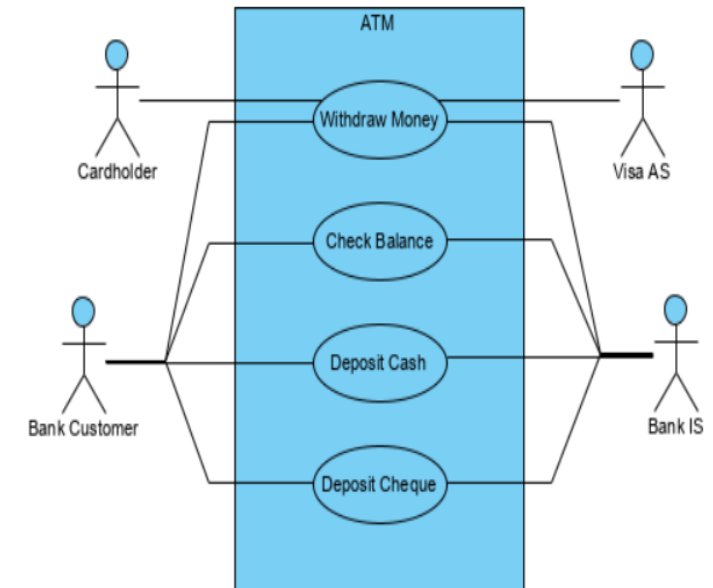
- Which user groups are supported by the system to perform their work?
- Which user groups execute the system's main functions?
- Which user groups perform secondary functions, such as maintenance and administration?
- With what external hardware or software system will the system interact?



The primary actor of a use case is the stakeholder that calls on the system to deliver one of its services.

A supporting actor (also known as a secondary actor) in a use case is an external actor that provides a service to the system under design.

In the example below, the Visa Card Holder and Bank Customer are Primary Actors, while the Visa AS and Bank IS are secondary actors.



Identifying Scenarios

- A scenario is “a narrative description of what people do and experience as they try to make use of computer systems and applications”.
- A scenario is a concrete, focused, informal description of a single feature of the system from the viewpoint of a single actor.
- Scenarios cannot (and are not intended to) replace use cases, as they focus on specific instances and concrete events (as opposed to complete and general descriptions).
- However, scenarios enhance requirements elicitation by providing a tool that is understandable to users and clients.

Scenarios can have many different uses during requirements elicitation and during other activities of the life cycle. Below is a selected number of scenario types taken from:

- **As-is scenarios** describe a current situation. During reengineering, for example, the current system is understood by observing users and describing their actions as scenarios. These scenarios can then be validated for correctness and accuracy with the users.
- **Visionary scenarios** describe a future system. Visionary scenarios are used both as a point in the modeling space by developers as they refine their ideas of the future system and as a communication medium to elicit requirements from users. Visionary scenarios can be viewed as an inexpensive prototype.
- **Evaluation scenarios** describe user tasks against which the system is to be evaluated. The collaborative development of evaluation scenarios by users and developers also improves the definition of the functionality tested by these scenarios.
- **Training scenarios** are tutorials used for introducing new users to the system. These are step-by-step instructions designed to hand-hold the user through common tasks.

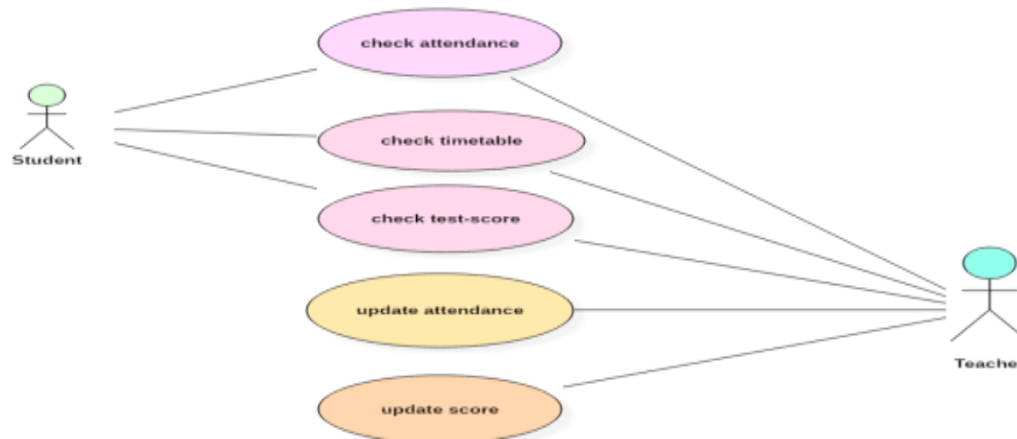
Questions for identifying scenarios

- What are the tasks that the actor wants the system to perform?
- What information does the actor access? Who creates that data? Can it be modified or removed? By whom?
- Which external changes does the actor need to inform the system about? How often? When?
- Which events does the system need to inform the actor about? With what latency?

Identifying Use Cases

A **scenario** is an instance of a **use case**; that is, a use case specifies all possible scenarios for a given piece of functionality. A use case is initiated by an actor. After its initiation, a use case may interact with other actors, as well. A use case represents a complete flow of events through the system in the sense that it describes a series of related interactions that result from its initiation.

Use cases are used to represent high-level functionalities and how the user will handle the system. A use case represents a distinct functionality of a system, a component, a package, or a class. It is denoted by an oval shape with the name of a use case written inside the oval shape.

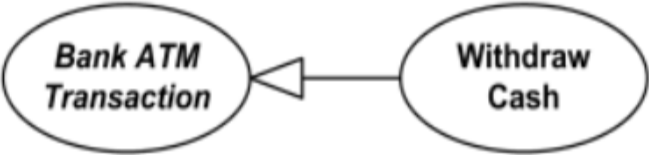
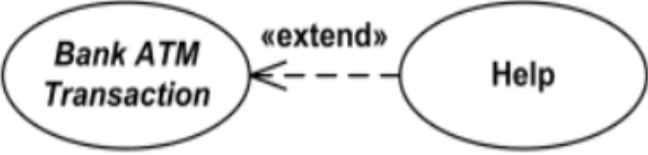
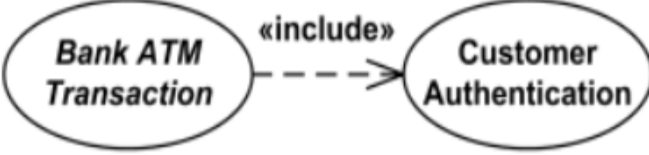


Simple Use Case Writing Guide

- Use cases should be named with verb phrases. The name of the use case should indicate what the user is trying to accomplish
- Actors should be named with noun phrases
- The boundary of the system should be clear. Steps accomplished by the actor and steps accomplished by the system should be distinguished
- Use case steps in the flow of events should be phrased in the active voice. This makes it explicit who accomplished the step.
- The causal relationship between successive steps should be clear.
- A use case should describe a complete user transaction.
- Exceptions should be described separately.
- A use case should not describe the user interface of the system. This takes away the focus from the actual steps accomplished by the user and is better addressed with visual mock-ups
- A use case should not exceed two or three pages in length. Otherwise, use include and extend relationships to decompose it in smaller use cases,

Identifying Relationships among Actors and Use Cases

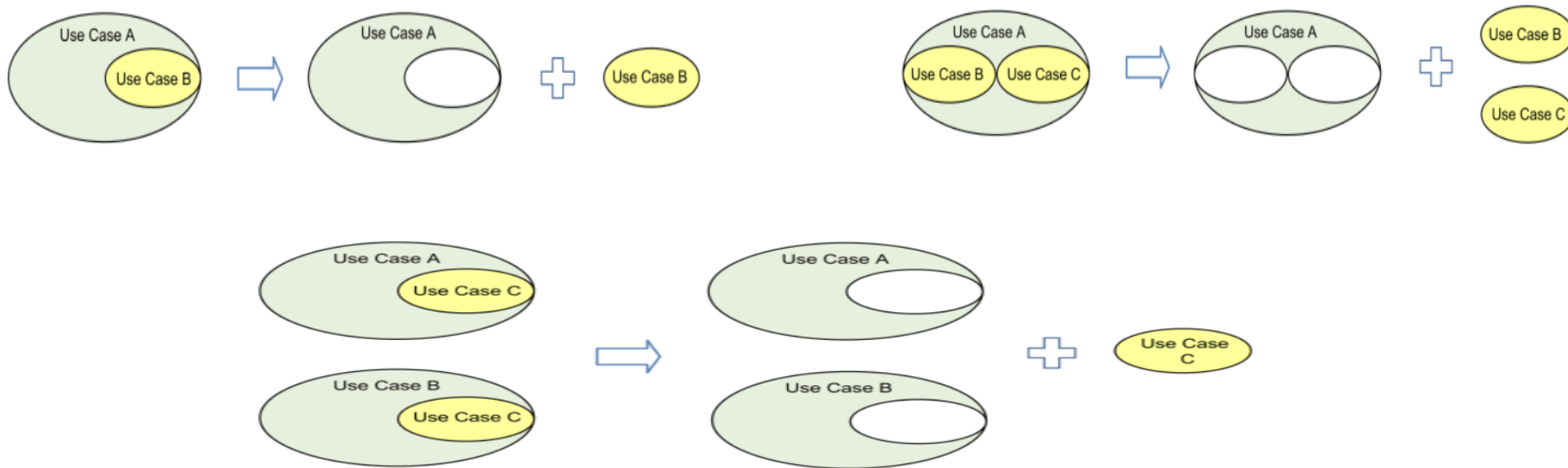
- Communication relationships between actors and use cases
- Extend relationships between use cases
- Include relationships between use cases
- Extend versus include relationships

Generalization	Extend	Include
		
Base use case could be abstract use case (incomplete) or concrete (complete).	Base use case is complete (concrete) by itself, defined independently.	Base use case is incomplete (abstract use case).
Specialized use case is required, not optional, if base use case is abstract.	Extending use case is optional, supplementary.	Included use case required, not optional.
No explicit location to use specialization.	Has at least one explicit extension location.	No explicit inclusion location but is included at some location.
No explicit condition to use specialization.	Could have optional extension condition.	No explicit inclusion condition.

Use case include is a directed relationship between two use cases which is used to show that behavior of the **included** use case (the addition) is inserted into the behavior of the **including** (the base) use case.

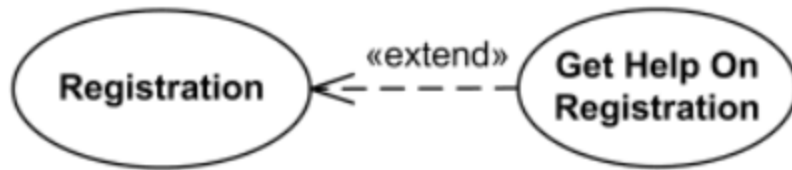
The **include** relationship could be used:

1. to simplify large use case by splitting it into several use cases,
2. to extract **common parts** of the behaviors of two or more use cases.

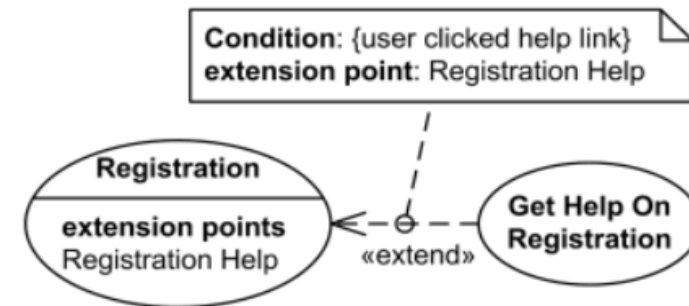


Extend is a directed relationship that specifies how and when the behavior defined in usually supplementary (optional) **extending use case** can be inserted into the behavior defined in the **extended use case**.

Extended use case is meaningful on its own, it is **independent** of the extending use case. **Extending** use case typically defines **optional** behavior that is not necessarily meaningful by itself. The extend relationship is **owned** by the extending use case. The same extending use case can extend more than one use case, and extending use case may itself be extended.

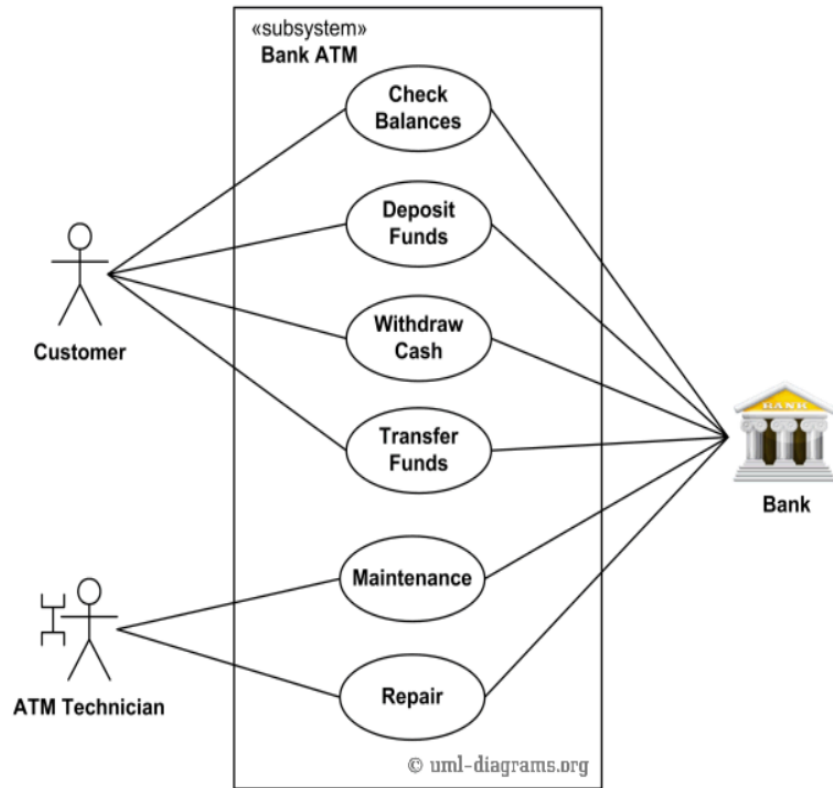


Registration use case is complete and meaningful on its own. It could be extended with optional **Get Help On Registration** use case.



Registration use case is conditionally extended by Get Help On Registration use case in extension point Registration Help.

Examples of use-case diagrams



Summary

- Use case diagrams are a way to capture the system's functionality and requirements in UML diagrams.
- It captures the dynamic behavior of a live system.
- A use case diagram consists of a use case and an actor.
- A use case represents a distinct functionality of a system, a component, a package, or a class.
- An actor is an entity that initiates the use case from outside the scope of a use case.
- The name of an actor or a use case must be meaningful and relevant to the system.
- A purpose of use case diagram is to capture the core functionalities of a system.

Key references

Bernd Bruegge & Allen H. Dutoit - Object-Oriented Software Engineering: Using UML, Patterns, and Java

<https://www.guru99.com/use-case-diagrams-example.html>

<https://www.uml-diagrams.org/use-case-include.html>

Thank You