Use Case Diagram Lecture Notes

USE CASE DIAGRAM-ELABORATION PHASE

Use case diagrams consist of 4 objects.
Actor
Use case
System
Package
Relation

By

Dr. Bandu B. Meshram
PhD(Computer Engineering), LLM(Constitutional law)
Computer Hacking and Forensic Investigator-EC Council Certified, USA

Professor and Former Head,
Department of Computer Engineering & Information Technology,
VJTI, Matunga, Mumbai-19

Use case diagram is a behavioral UML diagram type and frequently used to analyze various systems. They enable you to visualize the different types of roles in a system and how those roles interact with the system. This use case diagram tutorial will cover the following topics and help you create use cases better.

Importance of Use Case Diagrams

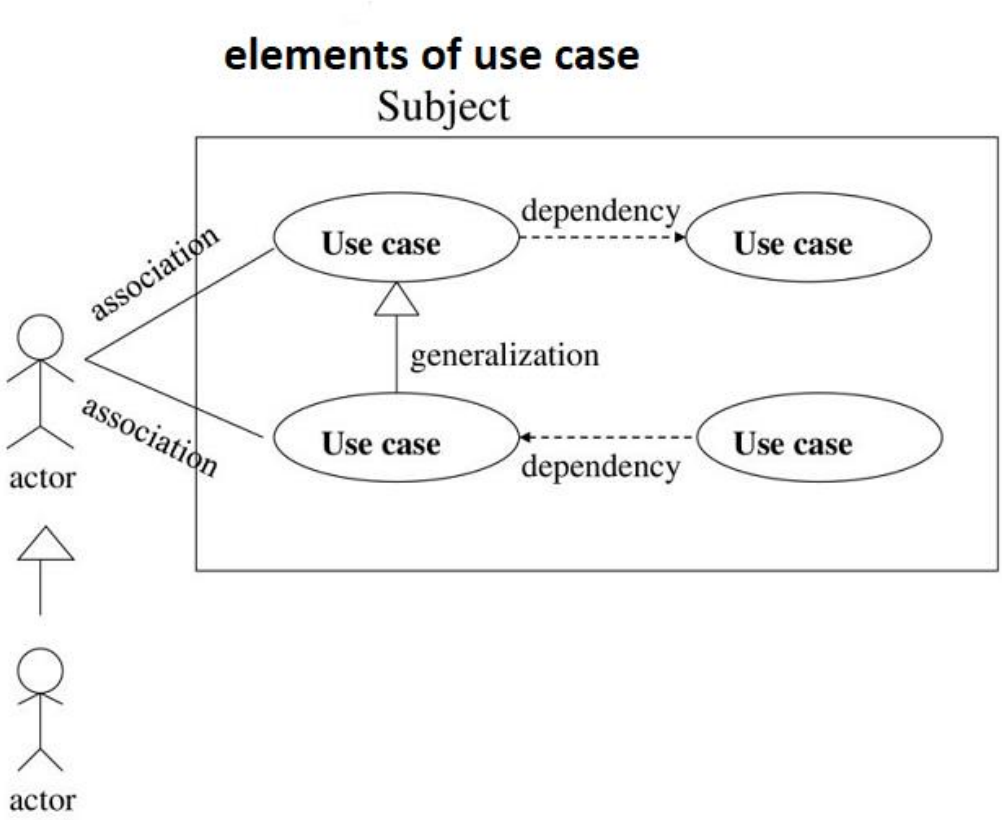
As mentioned before use case diagrams are used to gather a usage requirement of a system. Depending on your requirement you can use that data in different ways. Below are few ways to use them.

- **To identify functions and how roles interact with them** – The primary purpose of use case diagrams.
- **For a high-level view of the system** – Especially useful when presenting to managers or stakeholders. You can highlight the roles that interact with the system and the functionality provided by the system without going deep into inner workings of the system.
- **To identify internal and external factors** – This might sound simple but in large complex projects a system can be identified as an external role in another use case.

Use Case Diagram objects

Use case diagrams consist of 4 objects.

- Actor
- Use case
- System
- Package



The objects are further explained below.

Actor

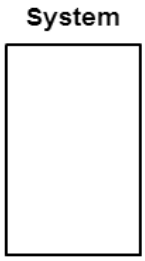
Actor in a use case diagram is **any entity that performs a role** in one given system. This could be a person, organization or an external system

Use Case

A use case **represents a function or an action within the system**. It's drawn as an oval and named with the function.

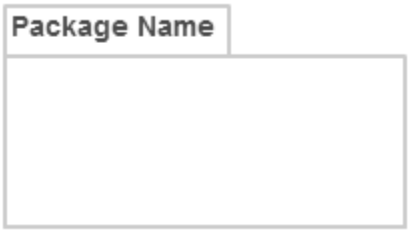
System

The system is used to **define the scope of the use case** and drawn as a rectangle. For example, you can create all the use cases and then use the system object to define the scope covered by your project.



Package

The package is another optional element that is extremely useful in complex diagrams. Similar to class diagrams, packages are **used to group together use cases**. They are drawn like the image shown below.



Use Case Diagram Guidelines

These include naming standards, directions of arrows, the placing of use cases, usage of system boxes and also proper usage of relationships.

Relationships in Use Case Diagrams

There are five types of relationships in a use case diagram. They are

- **Association between an actor and a use case**
- **Generalization of an actor**
- **Extend relationship between two use cases**

- **Include relationship between two use cases**
- **Generalization of a use case**

How to Create a Use Case Diagram

Up to now, you've learned about objects, relationships and guidelines that are critical when drawing use case diagrams. I'll explain the various processes using a banking system as an example.

Identifying Actors

Actors are external entities that interact with your system. **It can be a person, another system or an organization.**

In a banking system, the most obvious actor is the customer. Other actors can be bank employee or cashier depending on the role you're trying to show in the use case.

Primary actor and secondary actor

We know that the **primary actor** is the one that initiates a **use case** and a **secondary actor** is the one that helps completion of the **use case** through his specific support

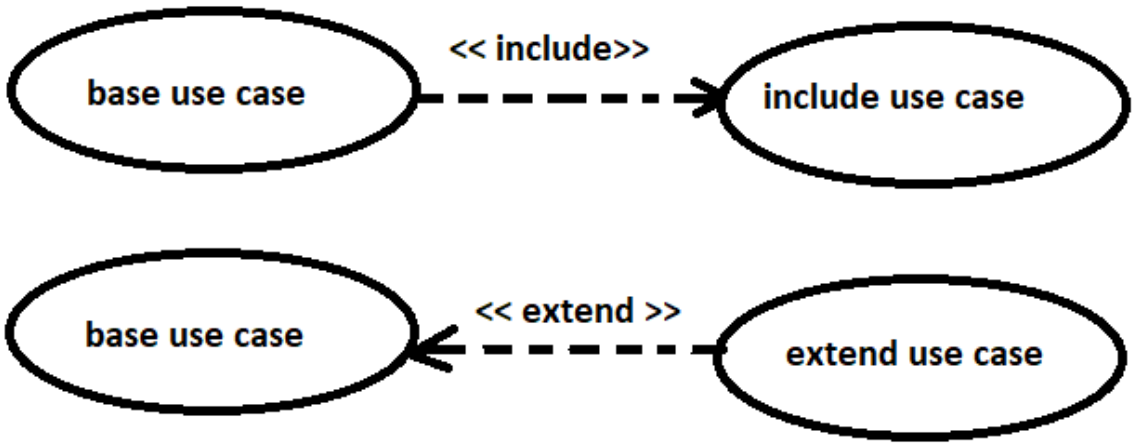
Identifying Use Cases

A good way to do this is to identify what the actors need from the system. In a banking system, a customer will need to open accounts, deposit and withdraw funds, request check books and similar functions. So all of these can be considered as use cases.

Look for Common Functionality to use Include

<< INCLUDE>>, OR << USES>>

If you find two or more use cases that share common functionality you can extract the common functions and add it to a separate use case. **Then you can connect it via the include relationship to show that it's always called when the original use case is executed**.



Optional Functions or Additional Functions

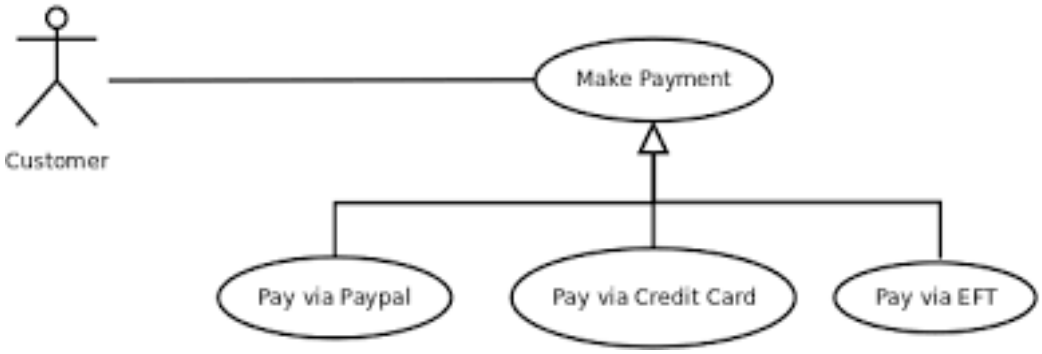
<< EXTEND>

Extend doesn't always mean it's optional.. **The thing to remember is that the base use case should be able to perform a function on its own even if the extending use case is not called.**

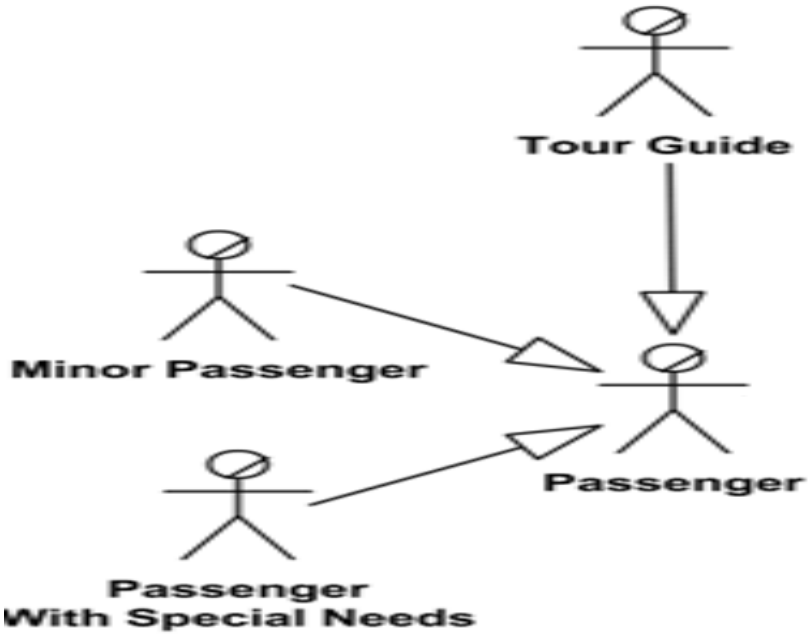
Is it Possible to Generalize Actors and Use Cases:yes

you can generalize the actor to show the inheritance of functions. You can do a similar thing for use case as well.

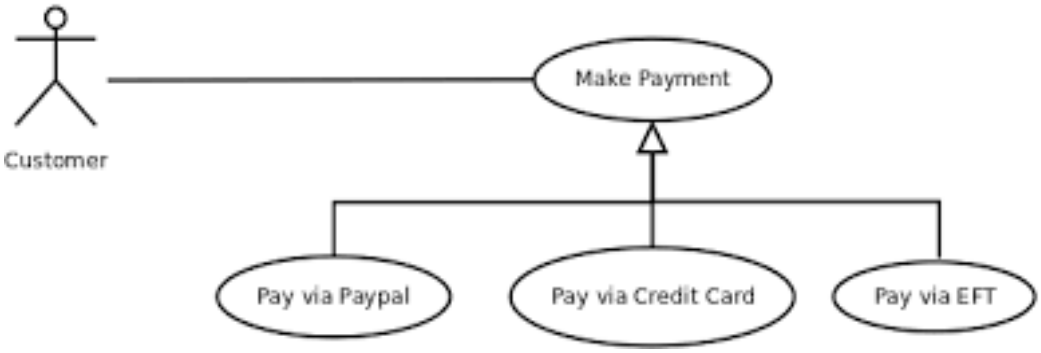
One of the best examples of this is "Make Payment" use case in a payment system. You can further generalize it to "Pay by Credit Card", "Pay by Cash", "Pay by Check" etc. All of them have the attributes and the functionality of payment with special scenarios unique to them.

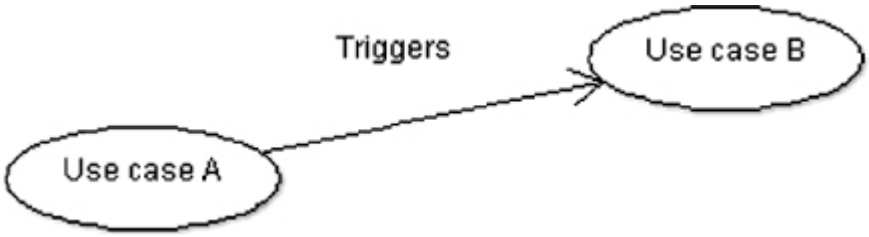


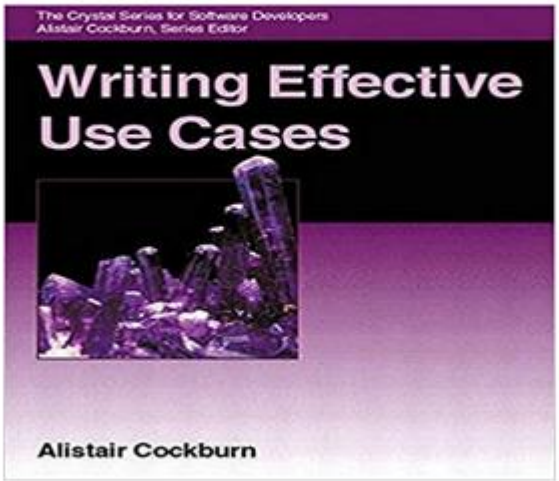
Generalization of actor



Generalization of use case




Triggers
Use case B
Use case A


The Crystal Series for Software Developers
Alistair Cockburn, Series Editor
Writing Effective Use Cases
Alistair Cockburn