UML Use Case Diagrams

Use case diagrams are usually referred to as behavior diagrams used to describe a set of actions (use cases) that some system or systems (subject) should or can perform in collaboration with one or more external users of the system (actors). Each use case should provide some observable and valuable result to the actors or other stakeholders of the system.

Note, that UML 2.0 to 2.4 specifications also described use case diagram as a specialization of a class diagram, and class diagram is a structure diagram.

Use case diagrams are in fact twofold - they are both behavior diagrams, because they describe behavior of the system, and they are also structure diagrams - as a special case of class diagrams where classifiers are restricted to be either actors or use cases related to each other with associations.

System Use Case Diagrams

(System) Use case diagrams are used to specify:

(external) requirements, required usages of a system under design or analysis (subject) - to capture what the system is supposed to do;
the functionality offered by a subject – what the system can do;
requirements the specified subject poses on its environment - by defining how environment should interact with the subject so that it will be able to perform its services.
Major elements of the UML use case diagram are shown on the picture below.

Business Use Case Diagrams

While support for business modeling was declared as one of the goals of the UML, UML specification provides no notation specific to business needs.

Business use cases were introduced in Rational Unified Process (RUP) to represent business function, process, or activity performed in the modeled business. A business actor represents a role played by some person or system external to the modeled business, and interacting with the business. Business use case should produce a result of observable value to a business actor.

Major elements of the business use case diagram are shown on the picture below. Note again, both business use case as well as business actor are not defined in UML standard, so you will either need to use some UML tool supporting those or create your own business modeling stereotypes.

Use Case Subject

A subject of a use case defines and represents boundaries of a business, software system, physical system or device, subsystem, component or even single class in relation to the requirements gathering and analysis.

In UML terms, subject is a classifier playing the "subject" role. They did not create a separate special class for subject, as it was done with actor and use case. UML 2.5 states that a subject of a UseCase could be a system or any other element that may have behavior, such as a Component or Class.". Classifier that may have owned behaviors is called behaviored classifier. Still, UML 2.5 abstract syntax shows subject as plain vanilla classifier.

A subject is a classifier (including subsystem, component, or even class) representing a business, software system, physical system or device under analysis, design, or consideration, having some behavior, and to which a set of use cases applies.

Subject is related to applicable use cases and is different from classifier owning use cases. In UML 2.x until 2.5 use case classifier was a classifier extended with the capability to own use cases. In UML 2.5 any classifier may own use cases.

Notation
Subject (sometimes called a system boundary) is presented by a rectangle with subject's name, associated keywords and stereotypes in the top left corner. Use cases applicable to the subject are located inside the rectangle and actors - outside of the system boundary.

Revisions
The requirement for the subject's name to be in the top left corner of the system boundary was added only in UML 2.5. Previous versions of UML allowed to use and had examples with the name in either one of top corners. Personally, I think it is too restrictive to use only the top left corner. When most of the actors are on the left side, subject's name looks better on the right side. In some cases I wouldn't mind having subject's name even in the top middle.

Software System Subject
System use cases describe a system that automates some business use case(s) or process. Subject in this case is software and/or hardware system, subsystem, component or device.

Examples of systems:

Web Site
Payment System
Automated Teller Machine (ATM)
Point of Sale (POS) Terminal

UML provides no standard stereotypes for subject (i.e. use case classifier) but UML 2.4 specification examples use the stereotypes from components:

«Subsystem»
«Process»
«Service»
«Component»

Applicability of Use Cases
Use cases visually located inside the system boundaries are use cases applicable to the subject (but not necessarily owned by the subject).It is possible for some use cases to be applicable to multiple subjects.

Ownership of Use Cases
In UML 2.4 subject could own some or all of applicable use cases. This owning (nesting) of a use case is represented using the standard notation for nested classifier.

UML Actor

An actor is behaviored classifier which specifies a role played by an external entity that interacts with the subject (e. g., by exchanging signals and data), a human user of the designed system, some other system or hardware using services of the subject.

The term "role" is used informally as some type, group or particular facet of users that require specific services from the subject modeled with associated use cases. When an external entity interacts with the subject, it plays the role of a specific actor. That single physical entity may play several different roles, and a specific role may be played by single or multiple different instances.

Since an actor is external to the subject, it is typically defined in the same classifier or package that incorporates the subject.

All actors must have names according to the assumed role. Examples of actor names (user roles):

Customer
Web Client
Student
Passenger
Payment System
Standard UML notation for actor is "stick man" icon with the name of the actor above or below of the icon. Actor names should follow the capitalization and punctuation guidelines for classes. The names of abstract actors should be shown in italics.
1.Custom icons that convey the kind of actor may also be used to denote an actor, such as using a separate icon(s) for non-human actors.
2.An actor may also be shown as a class rectangle with the standard keyword «actor», having usual notation for class compartments, if needed.
3.An actor can only have binary associations to use cases, components, and classes.

Business Actor
A business actor (introduced in Rational Unified Process (RUP) to support business modeling) represents a role played by some person or system external to the modeled business and interacting with the business. Note, business actor is not defined in UML standard.

Some typical examples of business actors are:

Customer
Supplier
Patron
Passenger
Authority
Bank
Each business actor represents something outside of the modeled business and should be involved with at least one business use case.

Business actor is represented in RUP by "stick man" icon with a line crossing its head.

We can define abstract or concrete actors and specialize them using generalization relationship.

Generalization between actors is rendered as a solid directed line with a large arrowhead (same as for generalization between classes).

UML Association

Between Actor and Use Case
Each use case represents a unit of useful functionality that subjects provide to actors. An association between an
actor and a use case indicates that the actor and the use case somehow interact or communicate with each other.

Only binary associations are allowed between actors and use cases.

An actor could be associated to one or several use cases.A use case may have one or several

associated actors.
If there are several actors associated to the same use case, it may not be obvious from use case diagram which actor
initiates the use case, i.e. is a "primary actor". (In non-standard UML, primary actors are those using system services, and supporting actors are actors providing services to the system.)

Multiplicity of Association Ends
UML allows the use of multiplicity at one or both ends of an association between the actor and the use case.

Multiplicity of a Use Case
When an actor has an association to a use case with a multiplicity that is greater than one at the use case end, it
means that a given actor can be involved in multiple use cases of that type.

UML 2.x specifications including UML 2.5 allow any reasonable interpretation of this kind of interaction of a given actor with multiple use cases, and intentionally keep it undefined.

For example, use case multiplicity could mean that an actor interacts with multiple use cases:
in parallel (concurrently), or
at different points in time (overlapping), or
mutually exclusive (sequentially, random, etc).
Multiplicity of an Actor
Required actor may be explicitly denoted using multiplicity 1 or greater. UML 2.5 also allows actor to be optional. Multiplicity 0..1 of actor means that the actor may or may not participate in any of their associated use cases.
When a use case has an association to an actor with a multiplicity that is greater than one at the actor end, it means
that more than one actor instance is involved in the use case.
UML 2.x specifications including UML 2.5 allow any reasonable interpretation of this kind of interaction of multiple actors with a use case, and intentionally keep it undefined.

For instance, multiplicity of actor could mean that interaction of a particular use case with several

separate actor
instances might be:

simultaneous (concurrent) interaction, or
overlapping interaction, at different points in time, or
mutually exclusive (sequential, random, etc.) interaction.

UML Use Case

Use cases allow to capture requirements of systems under design or consideration, describe functionality provided by those systems, and determine the requirements the systems pose on their environment.

UML Specifications, provide several slightly different definitions of use case. I compiled the definition below from those pieces.

Def：A use case is a kind of behaviored classifier that specifies a [complete] unit of [useful] functionality performed by [one or more] subjects to which the use case applies in collaboration with one or more actors, and which [for complete use cases] yields an observable result that is of some value to those actors [or other stakeholders] of each subject.

UML specifications require that "this functionality must always be completed for the UseCase to complete. It is deemed complete if, after its execution, the subject will be in a state in which no further inputs or actions are expected and the UseCase can be initiated again, or in an error state."

The problem with this requirement is that it doesn't consider extending use cases and included use cases. The extending use case may not necessarily be meaningful by itself. Included use case is some common part extracted to a separate use case. It is also seems inapplicable to require to yield an observable result.

It is also doubtful that use case functionality is always useful: "Each UseCase specifies a unit of useful functionality that the subject provides to its users."

"A UseCase may apply to any number of subjects." It is reasonable to expect to have at least one subject for each use case, otherwise the definition of use case will make no sense. At the same time, UseCase class description in UML specification allows use case to have no associated subjects.

While "the key concepts specified in this clause are Actors, UseCases, and subjects" one of the definitions of use case somehow also mentions "other stakeholders of the subject." It should be enough to have actors, and if "stakeholders" have something else to add, they should be included in the UML specification as a separate concept.

UML specifications until UML 2.5 required that use case functionality is initiated by an actor . In UML 2.5 this was removed, meaning that there could be some situations when system functionality is started by system itself while still providing useful result to an actor. For example, system could notify a customer that order was shipped, schedule user information cleanup and archiving, request some information from another system, etc.

Also, all UML 2.x specifications until UML 2.5 stated that use cases "are defined according to the needs of actors." This sentence was removed from UML 2.5 as some actors might have neither needs nor requirements by themselves.

Naming Use Cases

UML Specification provides no guidelines on use case names. The only requirement is that each use case must have a name. We should just follow use case definition to give some name to that unit of functionality performed by a system which provides some observable and useful result to an actor. Examples of use case names:

Make Purchase
Place Order
Update Subscription
Transfer Funds
Hire Employee
Manage Account

Notation

If a subject (or system boundary) is displayed, the use case ellipse is visually located inside the system boundary rectangle. Note, that this does not necessarily mean that the subject classifier owns the contained use cases, but merely that the use case applies to that classifier.

Use cases have no standard keywords or stereotypes. Use case could be shown with a custom stereotype above the name. As a classifier, use case has properties. A list of use case properties - operations and attributes - could be shown in a compartment within the use case oval below the use case name.

Abstract Use Case

All UML 2.x specifications including UML 2.5 do not mention, define or explain abstract use cases. UML 1.x specification mentioned that "the name of an abstract use case may be shown in italics" but since UML 2.0 this sentence was removed from UML specifications without any explanations.

One reason that the sentence was removed could be that because use case is a classifier, and any classifier could be abstract (with the name shown in italics), it is obvious that it should be applicable to the use cases as well.

On the other hand, as the sentence was removed and UML 2.5 does not mention abstract use cases at all and does not provide even a single example of abstract use cases, it could mean that they expect all use cases to be concrete, not abstract. In this case, it would be reasonable to have this situation explained explicitly in UML specification.

Assuming use case could be abstract and applying appropriate definition for the classifier, abstract use case is use case which does not have complete declaration (is incomplete) and (" typically", as UML specification says) can not be instantiated. An abstract use case is intended to be used by other use cases, e.g., as a target of generalization relationship. I hope that "the name of an abstract use case may be shown in italics" is still applicable in UML 2.5, as it was specified in UML 1.x.
When UML 2.4 specification describes include relationship between use cases, they explain that " what is left in a base use case is usually not complete", but for some reason avoiding to call it abstract use case. Generally, it should mean that including use case is always abstract.

Though UML specification avoids doing it, it is quite common to find sources that define including use cases as abstract use cases or essential use cases. While we may assume that including use cases are always abstract, included use case could probably be either abstract or concrete. Amazingly, there are some sources - that I can't agree with - providing exactly opposite explanation that including (base) use cases are "usually concrete", while included ("addition") use cases are "usually abstract".

To add even more to the confusion, yet other sources define abstract use cases as use cases described at the abstract level (business use cases, sometimes called essential use cases) as opposed to the system use cases.

Describing Use Case Behaviors :

Use case behaviors may be described in a natural language text (opaque behavior), which is current common practice, or by using UML behavior diagrams for specific behaviors such as

activity,
state machine,
interaction.
Behavior of a use case may also be described indirectly through a collaboration that uses the use case and its actors as the classifiers that type its parts.

Which of these techniques to use depends on the nature of the use case behavior as well as on the intended reader. These descriptions can be combined .

UML tools should allow linking behaviors to the described use case. Use case could be rendered in a frame labeled as use case or uc (abbreviated form). Content area of the frame could be represented by different kinds of UML diagrams describing behavior of the use case.

Relationships Between Use Cases

Use cases could be organized using following relationships:

generalization
association
extend
include

Generalization Between Use Cases

Generalization between use cases is similar to generalization between classes – child use case inherits properties and behavior of the parent use case and may override the behavior of the parent.

Generalization is shown as a solid directed line with a large hollow triangle arrowhead, the same as for generalization between classifiers, directed from the more specific use case to the general use case.

Generalization between use cases.
Web User Authentication use case is abstract use case
specialized by Login, Remember Me and Single Sign-On use cases.

Association Between Use Cases

Use cases can only be involved in binary associations.

Two use cases specifying the same subject cannot be associated since each of them individually describes a complete usage of the system.

UML Use Case Extend

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.

The extension takes place at one or more extension points defined in the extended use case.

Extend relationship is shown as a dashed line with an open arrowhead directed from the extending use case to the extended (base) use case. The arrow is labeled with the keyword «extend».
The condition of the extend relationship as well as the references to the extension points are optionally shown in a comment note attached to the corresponding extend relationship.

Extension Point

An extension point is a feature of a use case which identifies (references) a point in the behavior of the use case where that behavior can be extended by some other (extending) use case, as specified by extend relationship.

Extension points may be shown in a compartment of the use case oval symbol under the heading extension points. Each extension point must have a name, unique within a use case. Extension points are shown as a text string according to the syntax:

extension point ::= name [: explanation]

The optional explanation is some description usually given as informal text. It could be in other forms, such as the name of a state in a state machine, an activity in an activity diagram, some precondition or postcondition.
Extension points may be shown in a compartment of the use case rectangle with ellipse icon under

UML Use Case Include

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:

to simplify large use case by splitting it into several use cases,
to extract common parts of the behaviors of two or more use cases.
A large use case could have some behaviors which might be detached into distinct smaller use cases to be included back into the base use case using the UML include relationship. The purpose of this action is modularization of behaviors, making them more manageable.
When two or more use cases have some common behavior, this common part could be extracted into a separate use case to be included back by the use cases with the UML include relationship.
Execution of the included use case is analogous to a subroutine call or macro command in programming. All of the behavior of the included use case is executed at a single location in the including use case before execution of the including use case is resumed.

Note, while UML 2.x defines extension points for the extend relationship, there are no "inclusion points" to specify location or condition of inclusion for the include.

Including use case depends on the addition of the included use case, which is required and not optional. It means that including use case is not complete by itself, and so it would make sense to refer to the including use cases as abstract use cases. Neither of UML 2.x specifications up to the UML 2.5 even mentions abstract use cases. A number of other UML sources define abstract use case as including use case, while in fact it has to be the other way around: including use case is abstract use case. See discussion of the definition of abstract use cases.

Include relationship between use cases is shown by a dashed arrow with an open arrowhead from the including (base) use case to the included (common part) use case. The arrow is labeled with the keyword «include».
Large and complex Checkout use case has several use cases extracted, each smaller use case describing some logical unit of behavior. Note, that including Checkout use case becomes incomplete by itself and requires included use cases to be complete.

Use Case Relationships Compared
This site received many requests related to which use case relationship should be used in which situation. I combined several key points from UML 2.4 Specification into the table below. Also, take a look at related discussion in the next paragraph.

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

How to Draw UML Use Case Diagram

There are several approaches on how to draw UML use case diagrams. If you don't know from where to start, try to follow the steps described here.

1.Define Subject

Subject is a business, software system, subsystem, component, device, etc. that we are designing or just trying to understand how it is working. It is very important to define what kind of system it is, and what is its scope or boundary. Give it a proper name, and use appropriate stereotype, e.g. «Business» or «Subsystem».

By declaring a subject we are defining boundaries of the system, to be able to determine what or who is inside the system, and what or who is outside of it.

2.Define Actors

UML actor is some type, group or particular facet of users that require some services from the subject. Actor is an external entity, which could be a human user of the designed system, or some other system or device using our system.

3.Define Use Cases

Now as we defined the boundaries of the system that we are designing or analysing, and external users of the system, we need to define what do those users need from the system. Each use case specifies a unit of complete and useful functionality that the subject provides to the actor(s). Use case should reflect user needs and goals, and should be initiated by an actor.

4.Describe Use Case Behaviors

Use case behaviors may be described in a natural language text (opaque behavior), which is current common practice, or by using UML behavior diagrams for specific behaviors such as
activity,
state machine,
interaction.

UML tools should allow linking behaviors to the described use case. Example of such binding of a use case to the behavior represented by activity is shown below using UML 2.5 notation.

UML Use Case Diagram Example:Airport Check-In and Security Screening
This is an example of a business use case diagram which is usually created during Business Modeling and is rendered here in Rational Unified Process (RUP) notation.

Business actors are Passenger, Tour Guide, Minor (Child), Passenger with Special Needs (e.g. with disabilities), all playing external roles in relation to airport business.

Business use cases are Individual Check-In, Group Check-In (for groups of tourists), Security Screening, etc. - representing business functions or processes taking place in airport and serving the needs of passengers.

Business use cases Baggage Check-in and Baggage Handling extend Check-In use cases, because passenger might have no luggage, so baggage check-in and handling are optional.

UML Use Case Diagram Example:Restaurant
Here we provide two alternative examples of a business use case diagram for a Restaurant, rendered in a notation used by Rational Unified Process (RUP).

First example shows external business view of a restaurant. We can see several business actors having some needs and goals as related to the restaurant and business use cases expressing expectations of the actors from the business.
For example, Customer wants to Have Meal, Candidate - to Apply for Job, and Contractor - to fix some appliances. Note, that we don't have such actors as Chef or Waiter. They are not external roles but part of the business we model - the Restaurant, thus - they are not actors. In terms of RUP Chef and Waiter are business workers.

Second example shows internal business view of a restaurant. In this case we can see that restaurant has several business processes represented by business use cases which provide some services to external business actors. As in the previous example, actors have some needs and goals as related to the restaurant.

This approach could be more useful to model services that the business provides to different types of customers, but reading this kind of business use case diagrams could be confusing.

For example, Customer is now connected to Serve Meal use case, Supplier - to Purchase Supplies. We have now new actor Potential Customer participating in Advertise use case by reading ads and getting some information about restaurant. At the same time, Contractor actor is gone because Repair Appliances is not a service usually provided by restaurants.
Still, in this example we don't have actors as Chef or Waiter for the same reasons as before - they both are not external roles but part of the business we model.

UML Use Case Diagram Example:Online Shopping
Web Customer actor uses some web site to make purchases online. Top level use cases are View Items, Make Purchase and Client Register. View Items use case could be used by customer as top level use case if customer only wants to find and see some products. This use case could also be used as a part of Make Purchase use case. Client Register use case allows customer to register on the web site, for example to get some coupons or be invited to private sales. Note, that Checkout use case is included use case not available by itself - checkout is part of making purchase.

Except for the Web Customer actor there are several other actors which will be described below with detailed use cases.
View Items use case is extended by several optional use cases - customer may search for items, browse catalog, view items recommended for him/her, add items to shopping cart or wish list. All these use cases are extending use cases because they provide some optional functions allowing customer to find item.

Customer Authentication use case is included in View Recommended Items and Add to Wish List because both require the customer to be authenticated. At the same time, item could be added to the shopping cart without user authentication.

Checkout use case includes several required uses cases. Web customer should be authenticated. It could be done through user login page, user authentication cookie ("Remember me") or Single Sign-On (SSO). Web site authentication service is used in all these use cases, while SSO also requires participation of external identity provider.

Checkout use case also includes Payment use case which could be done either by using credit card and external credit payment service or with PayPal.
UML Use Case Diagram Example:Hospital Management
Hospital Management System is a large system including several subsystems or modules providing variety of functions. UML use case diagram example below shows actor and use cases for a hospital's reception.

Purpose: Describe major services (functionality) provided by a hospital's reception.

Hospital Reception subsystem or module supports some of the many job duties of hospital receptionist. Receptionist schedules patient' s appointments and admission to the hospital, collects information from patient upon patient's arrival and/or by phone. For the patient that will stay in the hospital ("inpatient") she or he should have a bed allotted in a ward. Receptionists might also receive patient's payments, record them in a database and provide receipts, file insurance claims and medical reports.

UML Use Case Diagram Example
This UML use case diagram example shows some use cases for a system which processes credit cards.

Credit Card Processing System (aka Credit Card Payment Gateway) is a subject, i.e. system under design or consideration. Primary actor for the system is a Merchant ’ s Credit Card Processing System. The merchant submits some credit card transaction request to the credit card payment gateway on behalf of a customer. Bank which issued customer's credit card is actor which could approve or reject the transaction. If transaction is approved, funds will be transferred to merchant's bank account.

Authorize and Capture use case is the most common type of credit card transaction. The requested amount of money should be first authorized by Customer's Credit Card Bank, and if approved, is further submitted for settlement. During the settlement funds approved for the credit card transaction are deposited into the Merchant's Bank account.

In some cases, only authorization is requested and the transaction will not be sent for settlement. In this case, usually if no further action is taken within some number of days, the authorization expires. Merchants can submit this request if they want to verify the availability of funds on the customer ’ s credit card, if item is not currently in stock, or if merchant wants to review orders before shipping.

Capture (request to capture funds that were previously authorized) use case describes several scenarios when merchant needs to complete some previously authorized transaction - either submitted through the payment gateway or requested without using the system, e.g. using voice authorization.

Credit use case describes situations when customer should receive a refund for a transaction that was either successfully processed and settled through the system or for some transaction that was not originally submitted through the payment gateway.

Void use case describes cases when it is needed to cancel one or several related transactions that were not yet settled. If possible, the transactions will not be sent for settlement. If the Void transaction fails, the original transaction is likely already settled.

Verify use case describes zero or small amount verification transactions which could also include verification of some client's data such as address.