Structuring Requirements: Use Case Descriptions and Diagrams

Use case modeling

Use case modeling is applied to analyze the **functional requirements** of a system. It is done in the early stages of the systems development, during the **analysis phase,** to help developers understand the functional requirements of the systems without worrying about how those requirements will be implemented.

The process is inherently **iterative**: developers need to involve the users in discussions throughout the model development process and finally come to agreement on the requirement specification.

What is a use case?

A use shows the behavior or the functionality of a system. It consists of a set of possible sequence of interaction between a system and a user in a particular environment and related to a goal.

A use case is a depiction of system's behavior or functionality under various conditions as the system respond to request from principal actors. A principal actor initiates a request and the system responds

A use case can be stated as what the system is **supposed to do** and **what the system is to act on.**

Examples;
-Enter sales data, compute commission, generate quarterly report, enter customer's details, compute account's balance, register students etc.

A use case driven design, in which the use cases controls the formation of all other models promotes requirements traceability among the different models used in OOSAD. **Requirements traceability,** is defined as "*the ability to describe and follow the life of a requirement in both a forwards and backwards direction (i.e., from its origins, through its development and specification, to its subsequent deployment and use, and through periods of ongoing refinement and iteration in any of these phases)*

If a user requirement changes during development life cycle, the changes are first made in the use case model. Changes in the use case model dictates what changes need to be done on other models. **Thus models are traceable.**

Traceability is an important aspect of system documentation as it allows the analyst to see how **different models are interconnected**, making later maintenance and the modification of the system easier.

A use-case model consists of **actors and use cases**.

An actor is an external entity that interacts with the system. It is someone or something that exchanges information with the system.

A use case represents a sequence of related actions initiated by an actor; it is a specific way of using the system. Note that a user and actor are not the same.

A user is anyone who uses the system while the actor represents a role that a user can play. The actor's name should indicate that role. Actors help you to identify the use cases they carry out.

Use-case diagram:

Graphical illustrations of use cases and how they are related to each other. It's a diagram/picture that depicts the use cases and actors for a system.

In UML, a use-case diagram is depicted diagrammatically as a **box** inside which are use cases represented as **ellipses** (with their names underneath) and outside are actors. An actor is shown using a **stick figure** with its name below

<u>Example 1</u> Consider a use-case diagram is for a *university registration system*. Such a system would have the following use cases:

Class registration, Registration for special class, Prerequisite courses not completed, and Student billing

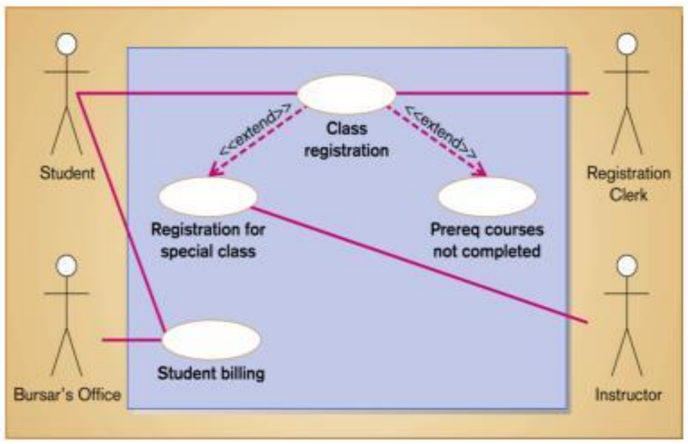
These use cases, then, represent the typical interactions the system has with its users. These use cases are performed by the actors outside the system which would include the following:

Student, Registration clerk, Instructor, and Bursar's office—that interact with the system (shown by the lines touching the actors).

The use case diagram for such a system could the drawn as follows;

point

Use-case diagram for a university registration system



correctiondotted arrow points upwards instead of downwards***

A use case is always initiated by an actor. For example, Student billing is initiated by the Bursar's office. A use case can interact with actors other than the one that initiated it. The Student billing use case, although initiated by the Bursar's office, interacts with the Students by mailing them tuition invoices.

Another use case, Class registration, is carried out by two actors, Student and Registration clerk. This use case performs a series of **related actions** aimed at registering a student for a class.

A use case represents a **complete functionality**. You should not represent an individual action that is part of an overall function as a use case. For example, although submitting a registration form and paying tuition are two actions performed by users (students) in the university registration system, we do not show them as use cases, because they do not specify a complete course of events; each of these actions is executed only as part of an overall function or use case. You can think of "Submit registration form" as one of the actions of the Class registration use case, and "Pay tuition" as one of the actions of the Student billing use case.

A use case may participate in ***relationships*** with other use cases. An ***extends relationship***, as a line with a hollow triangle pointing toward the extended use case and labeled with the "< extend>" symbol, extends a use case by adding new behaviors or actions.

Example the Registration for special class use case **extends** the Class registration use case by capturing the additional actions that need to be performed in registering a student for a special class. Registering for a special class requires prior permission of the instructor, in addition to the other steps carried out for a regular registration.

Another example of an extends relationship is that between the Prerequisite courses not completed and Class registration use cases. The former **extends** the latter in situations where a student registering for a class has not taken the prerequisite courses.

Extends relationship -It is a association between two use cases where one use case adds new behavior or functionality to the other

Include relationship.

Another kind of relationship is ***include***, which arises when one use case references another use case. An include relationship is also shown diagrammatically as a dashed line with a hollow arrowhead pointing toward the use case that is being used; the line is labeled with the "<include>" symbol.

The relationship implies that the use case where the arrow originates uses the use case where the arrow ends while it is executing. Typically, the use case that is included represents a generic function that is common to many business function. Rather than reproduce that functionality within every use case that needs it, the functionality is factored into a separate use case that can then be used by others use cases.

Include relationship-association between two use cases where one use case uses the functionality contained in the other.

<u>**Example 2:**</u> a use case diagram for any business that needs to **reorder supplies** on a regular basis such as **a restaurant.**

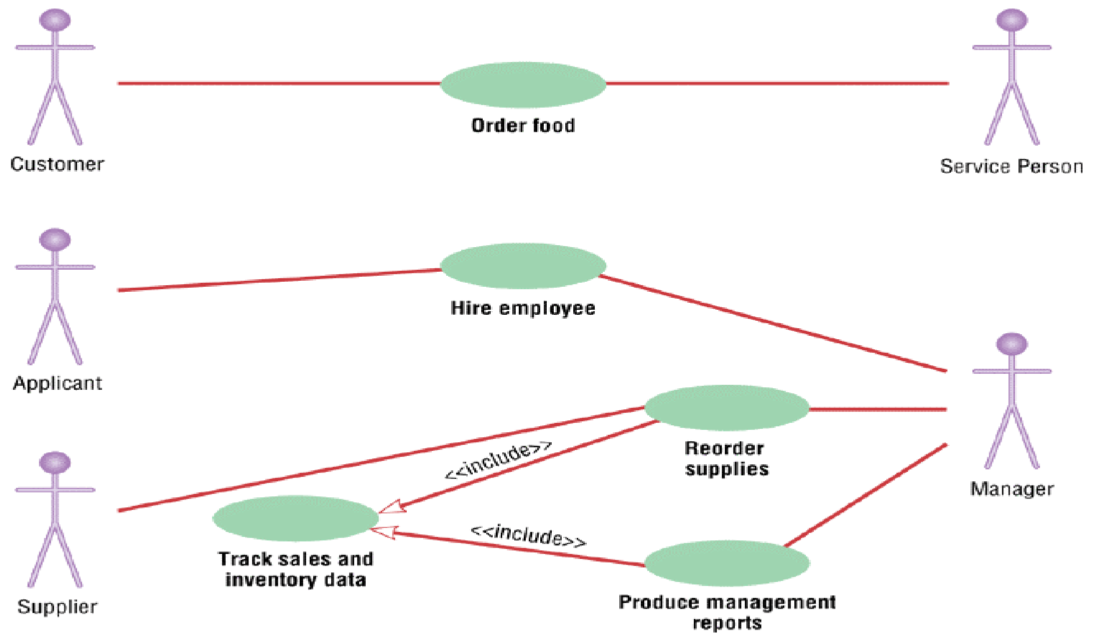
Such a system would have the following use cases:

Order food, hire employee, reorder supplies, Track sales and inventory data, produce management report

Actors would include the following: ***Customer, applicant, manager, supplier service person***

Thus the use case diagram would be drawn as follows;

Figure A.2 Use-Case Diagram for a Hoosier Burger System



The Customer actor initiates the *Order food* use case; the other actor involved is the Service Person. A specific scenario would represent a customer placing an order with a service person.

Reorder supplies use case involves the manager and supplier actors. The manager initiates the use case which then sends requests to supplier for various items.

The *include relationship* between the *Reorder supplies* and *Track sales and inventory data* use cases implies that the former uses the latter while executing. Simply put, when a manager reorders supplies, the sales and inventory data are tracked. The same data are also tracked when management reports are produced, so there is another include relationship between the Produce management reports and Track sales and inventory data use cases.

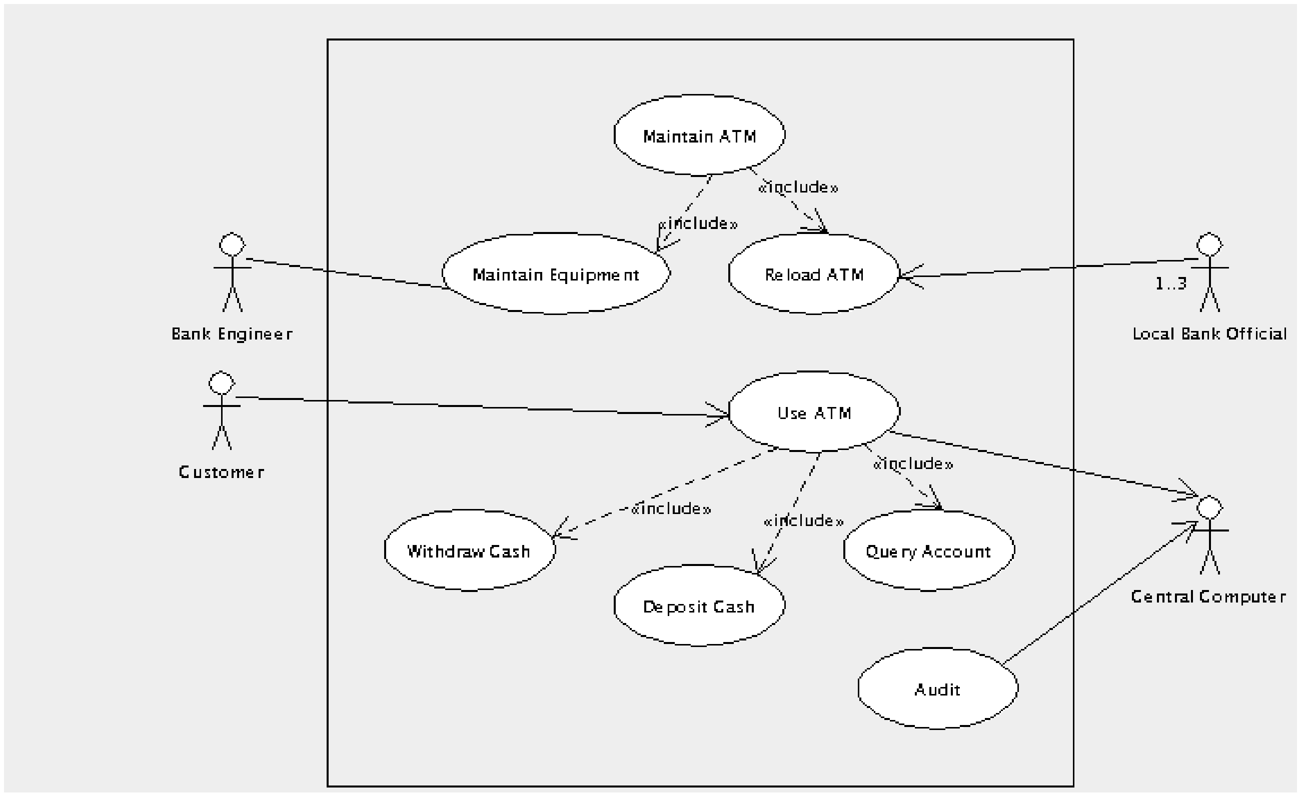
The Track sales and inventory data is a **generalized use case**, representing the common behavior among the **specialized use cases**, Reorder supplies and Produce management reports. When Reorder supplies or Produce management reports is performed, the entire Track sales and inventory data is used.

-Such a use case (Track sales and inventory data) that is never executed on itself is called an **abstract** use case

NB: There is no connection between the actor and the abstract use case

More on use case diagrams

<u>Example 3: Banking System Use Case Diagram</u>



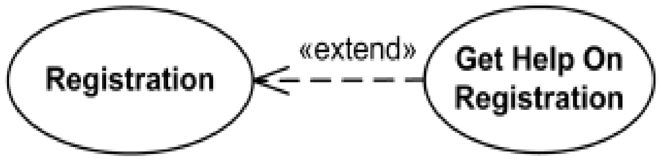
Extend is a directed relationship that specifies how and when the behavior defined in usually supplementary (optional) **extending use case** can be inserted into the behavior defined in the **extended use case**.

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»**.



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

<u>Note</u>

Include is used to extract use case fragments that are *duplicated* in multiple use cases. The included use case cannot stand alone and the original use case is not complete without the included one. This should be used sparingly an only in cases where the duplication is significant and exists by design (rather than by coincidence).

For example, the flow of events that occurs at the beginning of every ATM use case (when the user puts in their ATM card, enters their PIN, and is shown the main menu) would be a good candidate for an **include**.

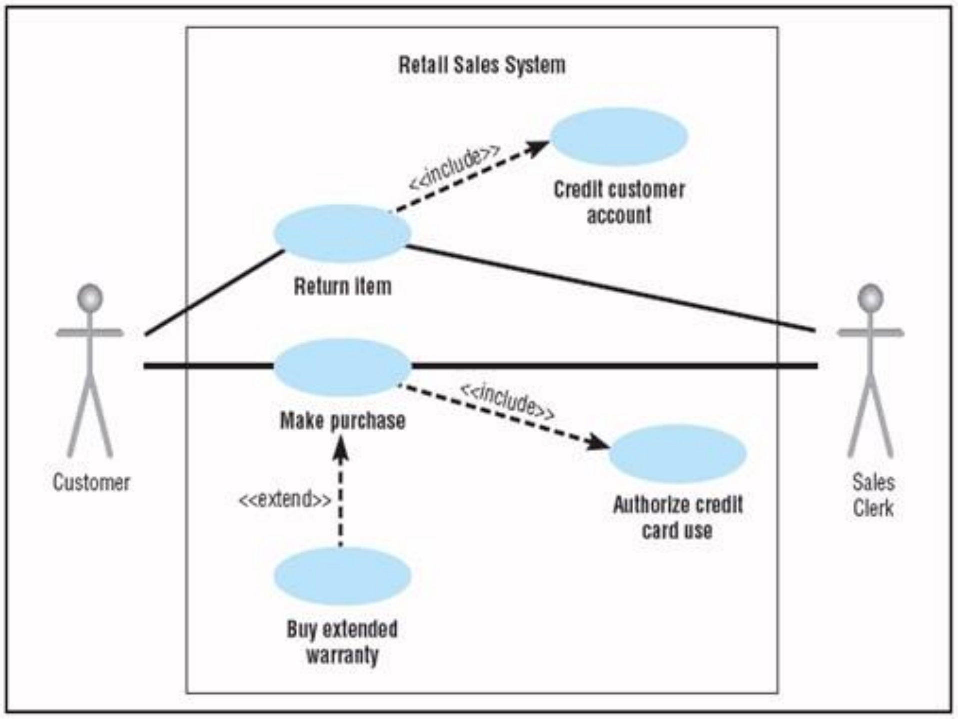
Extend is used when a use case conditionally adds steps to another first class use case.

<u>Example 4: POS retail sales system like in a supermarket</u>

Actors: **customer and the sales clerk**
Use cases:
i)Two specialized use cases i.e. **Return item and make purchase** each initiated by the customer actor

ii) Three abstract use cases i.e. **Authorize credit card use, credit customer account, buy extended warranty**

Figure 6.6 A Use-Case Diagram of a Retail Sales System, Featuring Insert Relations and an Extend Relation



When a customer actor initiates return item use case he expects a new item in exchange, cash refund or a credit on customer credit card account if the item was paid through credit card. This constitutes an **include relationship.**

Another include relationship is between make purchase and authorize credit case use. Every time a customer makes a purchase with a credit card, authorize credit case use is used too. The store where the purchase is made, makes a connection to credit card issuer to get authorization to charge the purchase to customer’s card. If the customer has maxed out his account, the store won’t be able to charge the account.
An **extend relationship** also exists between make purchase and buy extended warranty. Every time an electronic item is bought sales clerk requests the customer to buy extended warranty to protect the item after manufacturer warranty expires.

Creating such an extended warranty would not occur for every item sold as such it would not be an integral part of **make purchase** use case. “Buy extended warranty” does, however, expand the functionality of make purchase without changing the use case.