USE CASES

- use case is a description of a set of sequences of actions, including variants, that a system performs to yield an observable result of value to an actor
- use case captures the intended behavior of the system (or subsystem, class, or interface) without having to specify how that behavior is implemented
- a use case is rendered(represented) as an ellipse Figure 1: Actors and Use Cases
- Every use case must have a name that distinguishes it from other use cases: simple name and path name Figure 2: Simple and Path Names

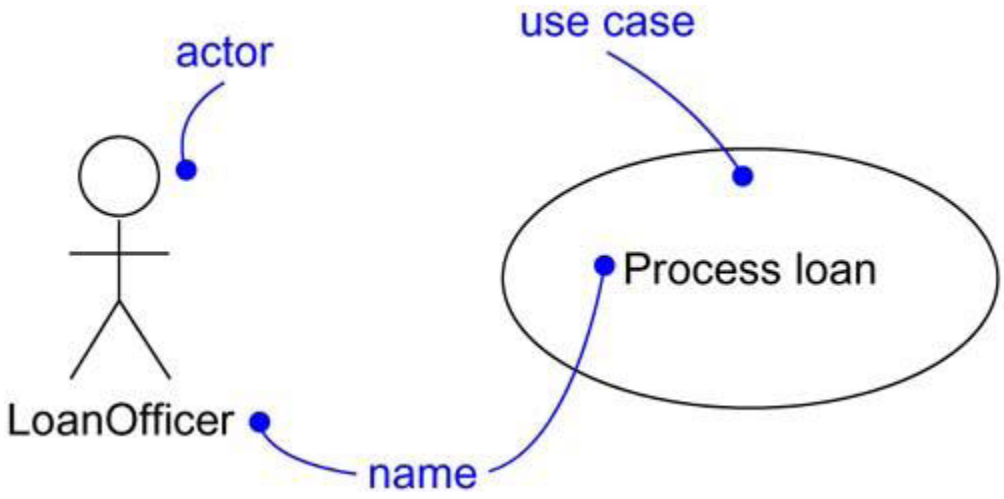

Figure 1: Actors and Use Cases

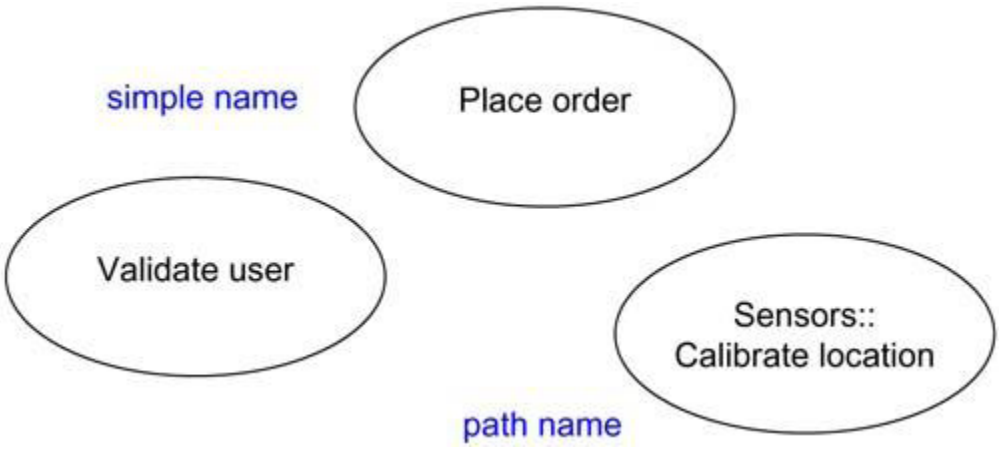

Figure 2: Simple and Path Names

Actors

- actor represents a coherent set of roles that users of use cases play when interacting with these use cases
- an actor represents a role that a human, a hardware device, or even another system plays with a system Figure 3: Actors

Actors may be connected to use cases by association

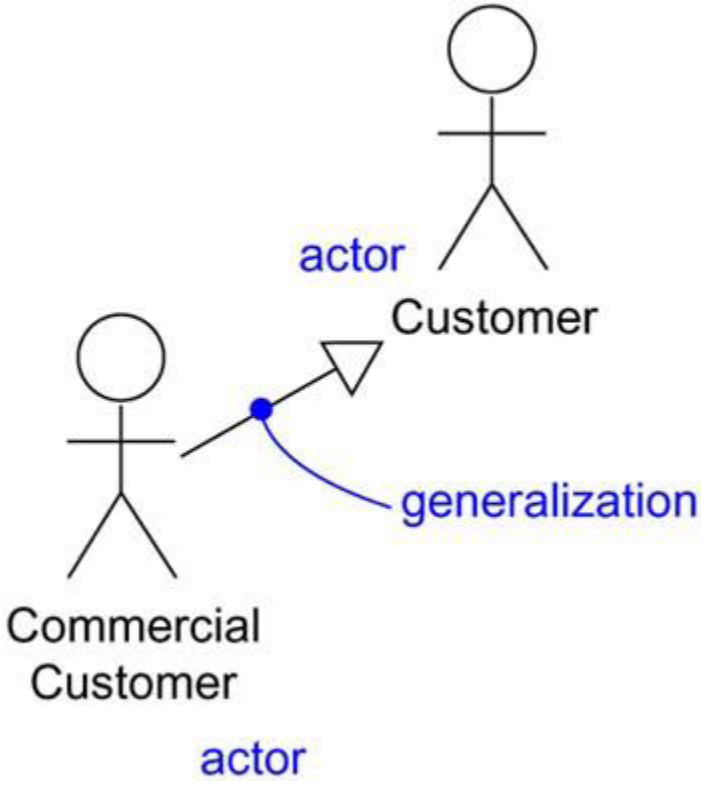

Figure 3: Actors

Use Cases & Flow of Events

flow of events include how and when the use case starts and ends when the use case interacts with the actors and what objects are exchanged, and the basic flow and alternative flows of the behavior.

The behavior of a use case can be specified by describing a flow of events in text.

There can be Main flow of events and one or more Exceptional flow of events.

Use Cases and Scenarios

- A scenario is a specific sequence of actions that illustrates behavior

- Scenarios are to use cases, as instances are to classes means that scenario is basically one instance of a use case
- for each use case, there will be primary scenarios and secondary scenarios.

Use Cases and Collaborations

- Collaborations are used to implement the behaviour of use cases with society of classes and other elements that work together
- It includes static and dynamic structure
- Figure 4: Use Cases and Collaborations

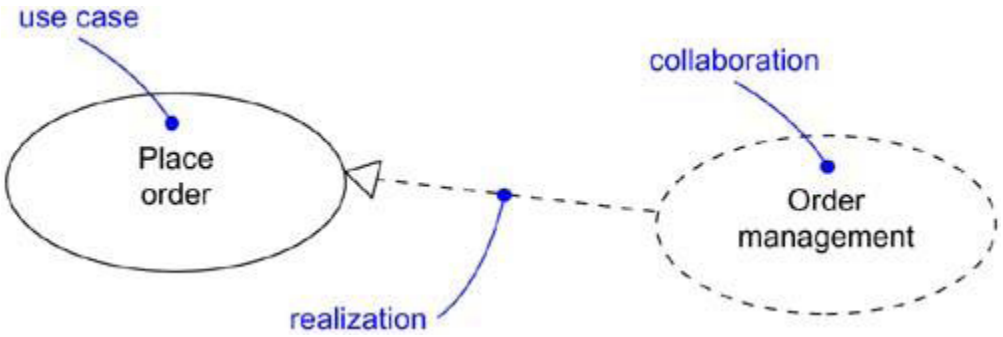

Figure 4: Use Cases and Collaborations

Organizing Use Cases

- organize use cases by grouping them in packages
- organize use cases by specifying generalization, include, and extend relationships among them

Generalization, Include and Extend

An *include relationship* between use cases means that the base use case explicitly incorporates the behavior of another use case at a location specified in the base.
An include relationship can be rendered as a dependency, stereotyped as include

An *extend relationship* between use cases means that the base use case implicitly incorporates the behavior of another use case at a location specified indirectly by the extending use case
An extend relationship can be rendered as a dependency, stereotyped as extend.
Figure:5

extension points are just labels that may appear in the flow of the base use case

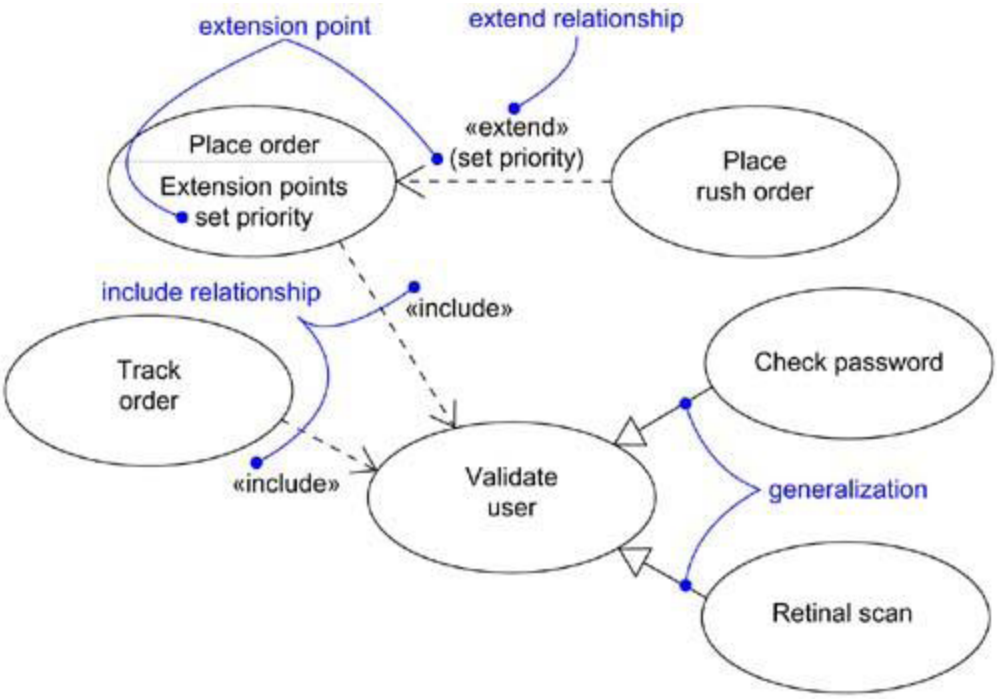

Figure 5: Generalization, Include and Extend

Modeling the Behavior of an Element

To model the behavior of an element,

- Identify the actors that interact with the element Candidate actors include groups that require certain behavior to perform their tasks or that are needed directly or indirectly to perform the element's functions
- Organize actors by identifying general and more specialized roles
- For each actor, consider the primary ways in which that actor interacts with the element Consider also interactions that change the state of the element or its environment or that involve a response to some event
- Consider also the exceptional ways in which each actor interacts with the element
- Organize these behaviors as use cases, applying include and extend

Figure 6: shows Modeling the Behavior of an Element

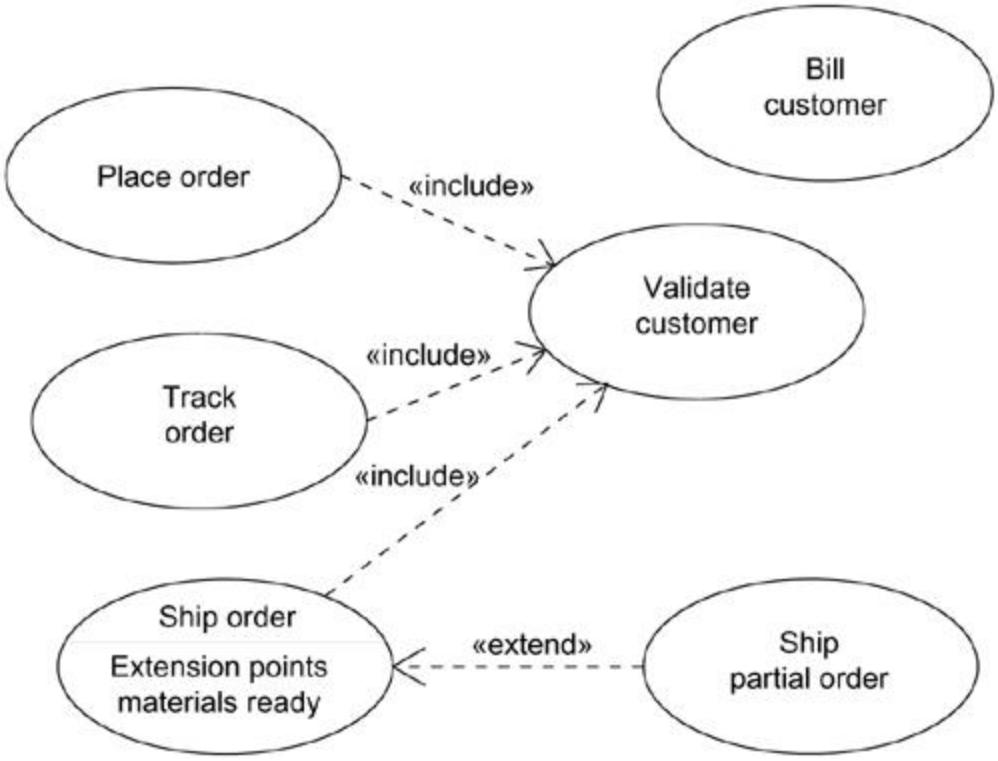

Figure 6: Modeling the Behavior of an Element

Source : http://praveenthomasln.wordpress.com/2012/04/05/use-cases-s8-cs/