



**SYSTEM USE CASE MODELING
STANDARDS AND GUIDELINES
VERSION 1.0**

DECEMBER 2, 2005

Information and Technology Management Branch

❖ IM / IT Standards & Guidelines ❖

Table of contents

DOCUMENT PURPOSE	3
DIAGRAM PURPOSE	3
DIAGRAM ELEMENTS	3
ACTORS	3
USE CASES	4
ASSOCIATIONS.....	4
NOTE	7
SYSTEM BOUNDARY (OPTIONAL)	8
PACKAGES	9
NAMING STANDARDS	9
ACTOR NAMING	9
USE CASE NAMING.....	9
ASSOCIATION NAMING	9
PACKAGE NAMING	10
MODELING STANDARDS	10
ACTOR MODELING	10
USE CASE MODELING	10
ASSOCIATION MODELING.....	11
NOTE MODELING	12
SYSTEM BOUNDARY MODELING (OPTIONAL).....	12
PACKAGE MODELING	12
SPECIFICATION STANDARDS	14

Document Purpose

The purpose of this document is to provide a set of minimal acceptable standards and guidelines for modeling the *System Use cases* based on UML 2.0. The document briefly describes the elements of a typical *System Use case diagram* followed by naming and modeling standards for the elements. The document assumes that the reader has prior knowledge of UML and Use case modeling in particular. The document does not address the best practices of use case modeling and writing.

Diagram Purpose

In general, a use case model is comprised of one or more *use case diagrams* and other supporting documentation in narrative style such as *use case specifications*. [An *Activity Diagram* serving as additional supporting diagram may also be created for each individual use case to depict the detailed logic of the individual use cases. *Activity Diagram* standards are available in a separate document]. Each use case typically has one basic course of action and one or more alternate courses of actions. The basic course of action is the main start-to-finish path that the use case will follow, where as the alternate courses represent the infrequently used paths and exceptions, error conditions etc. The basic and alternate courses of action of a use case are not depicted in the use case diagram but they are documented in narrative style in the use case specifications.

System use cases are used to model the application's requirements with focus on applications' target technology and implementation details. Consequently, the system use cases reflect design decisions in addition to requirements and also the references to system objects such as screens, reports, interfaces etc. The system use case has many high-level implementation details embedded within it.

System Use cases should achieve the following goals :

- Serve as an effective communication tool between analysts and developers.
- Is diagrammed using the standards described in this document.
- *Be documented in narrative style using the standard "System Use case Specification Template".*

Diagram elements

A System Use case diagram is comprised of the following diagram elements.

Actors


An *Actor* is a person, organization or an external system/sub-system/program that has interactions with the Application. Actors, by definition, are external to the system with which they are having interactions. Actors have goals that are achieved by use cases. Typically, Actors have behavior and are represented by the roles they play in the use cases. An Actor

stimulates the system by providing input and/or receiving something of measurable value from the system.

Broadly there are three types of actors in a System Use case model :

- Stakeholders - Actors having a vested interest in the behavior of the use case. Some stakeholders do not interact with the system even though they have interest in the system's behavior.
- Primary Actors - Actors who often trigger the use case.
- Supporting Actors - Actors who interact with the use case but do not trigger it.

Actor profiles and their roles within the system are not depicted in the use case diagram - rather these are documented in the use case specifications.


Actor is typically drawn as a *stick man shape* : .

It is also possible (and desirable in some situations) to draw the actor in other shapes using "stereotypes" feature of the UML tool.

Standards pertaining to Actor naming and modeling are described in later sections in this document.

Use Cases

A use case describes a sequence of actions that provide something of measurable value to an actor. A use case is a narrative that describes the sequence of events (including the variants) of an actor using a system to complete a process. Dr. Alistair Cockburn, one of the industry's use case gurus describes use cases as “sequences of interactions between the system and its external actors, related to a particular goal”. Use cases help capture the intended behavior of the system being developed.

Use case is always drawn as a *horizontal ellipse* : .

Standards pertaining to Use case naming and modeling are described in later sections in this document.

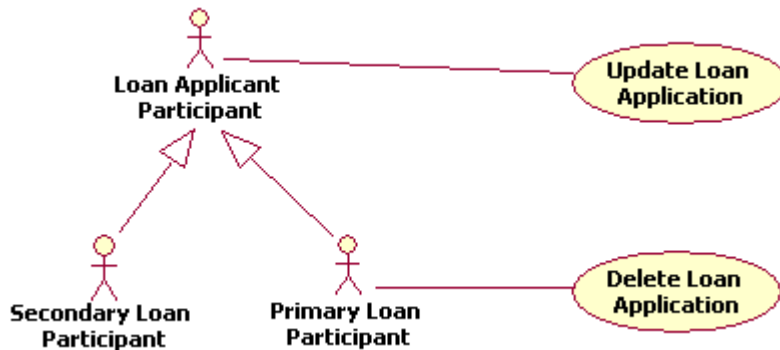
Associations

If an actor is involved with the use case by way of triggering the use case, supplying information to the use case, receiving information from the use case or in any other manner, then an association should be drawn between the Actor and the use case. There are three types of associations in a use case diagram and these are described below.

Association between Actor and Use case : Associations between actors and use cases is the most common type of association in a use case diagram. This association is drawn as a **solid line without arrow heads** (—) as shown in the example below. No label is attached to this type of association.



Association between Actors (generalization or inheritance) : Actor to Actor association is drawn to generalize the actor and depict both the general and specialized interactions of the Actors with the use case. Actor generalization is drawn as a **solid line with a closed arrowhead** (—▷) pointing towards the generalized actor, as shown in the example below. No label is attached to this type of association.

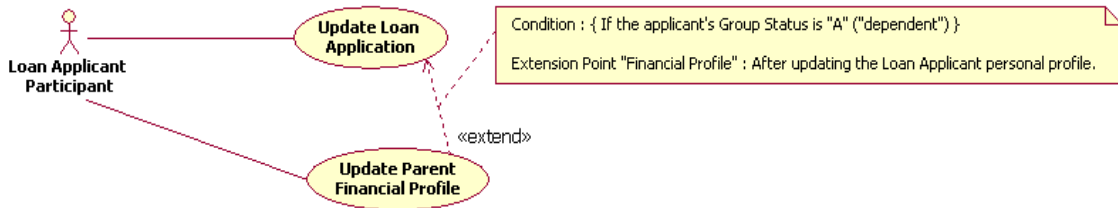


Please refer and use "Actor Profile Specification Template" to specify Actor generalizations. The Actor Profile Specification indicates how each Actor accesses the system, what access/security levels are needed etc.

Association between Use cases (reuse) : Use case to Use case association is drawn to depict the reuse of one Use cases' logic in another use case. The association between two use cases is depicted as a **broken line with an open arrowhead** (--->) pointing towards the base use case.

<<extend>> association : If the second use case is continuing (extending) the behavior of the first (base) use case under a specific condition, then the association is labeled as <<extend>> with the arrowhead pointing towards base use case as shown in the example below. Extension is a way of capturing a variant to a use case. The extending use case conceptually inserts the additional action sequences into the base use-case sequence. This allows the extending use case to continue the activity sequence of a base use case when the appropriate extension point is reached in the

base use case and the extension condition is fulfilled. The extension point (and the actual condition in the base use case under which the extended use case is to be executed) is documented in the use case diagram in the form of "Note" element.



<<include>> association : If the second use case is invoking (including) the behavior of the first (base) use case unconditionally, then the association is labeled as <<include>> with the arrowhead pointing towards the included (lower level) use case as shown in the example below. In general the included use case will be invoked every time the base use case is executed. The <<includes>> association helps reduce the duplication of functionality by factoring out common behavior into Use cases that are re-used.

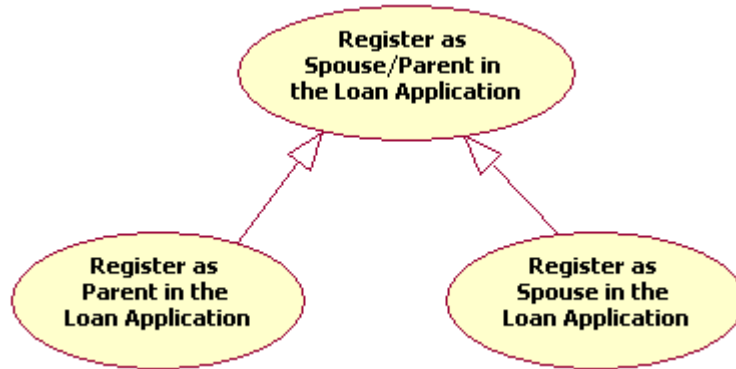


Unlike the "extension point", there is no explicit "inclusion point" shown on the use case diagram. Inclusion points are documented in the "Use case specification template" by mentioning "include <<use case name>>" at the appropriate step in the basic or alternate flow of events as shown in the example shown below :

Use Case: Submit Online Loan Application	
Actors:	Loan Applicant Participant
Precondition:	1. The Loan Applicant has browsed to the Loan Application web page.
Basic flow of events:	1. The Loan Applicant types some basic personal information as prompted by the web page and then clicks "Determine Loan Eligibility" button. 2. include("Pre-Screen Applicant" Use case) 3. The System displays if and why the Loan Applicant is not eligible to submit Online Loan Application.
Postcondition:
Alternate flow:

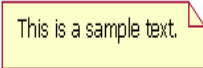
"inherit" association : It is possible for the second use case to simply "inherit" the behavior of the first use case without "extending" or "including". The inheriting use case would completely replace one or more of the courses of action of the inherited

use case. The inheriting (child) use case will contain all the attributes, sequences of behavior, and the extension points defined in the inherited (parent) use case, and participate in all the relationships of the inherited (parent) use case. In this case the association is not labeled and it is drawn as a solid line with a closed arrowhead (\rightarrow), as shown in the example below.



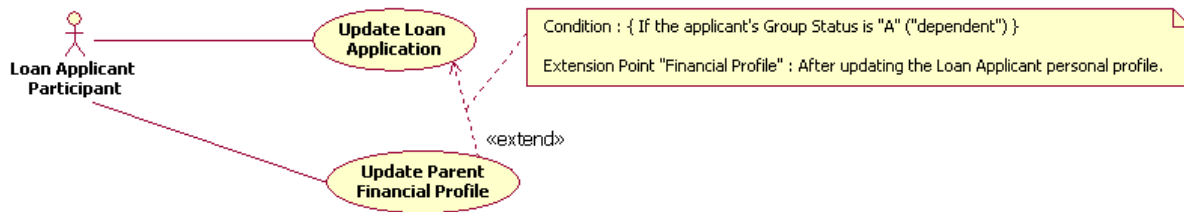
Note

A note is a model element that contains a comment or other textual information. In a use case diagram, a note is often connected to an Actor, Use case or Association in order to provide additional description or comments about that element that cannot be represented diagrammatically in the model.



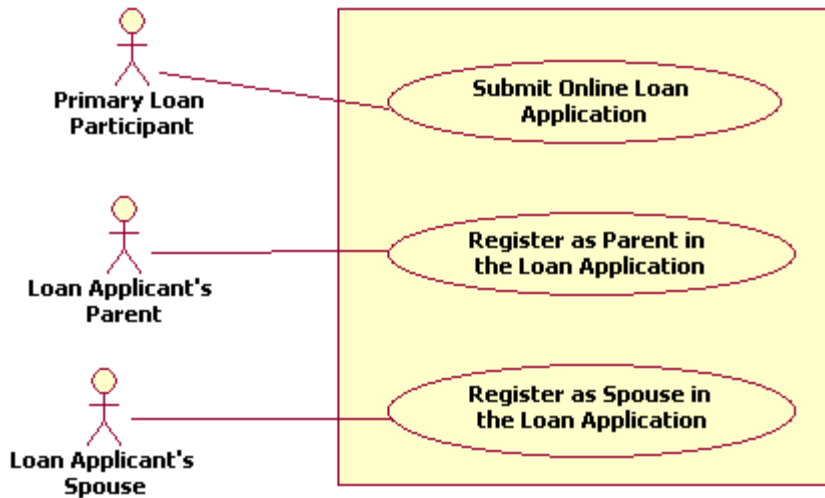
A note is always drawn as a rectangle with its upper-right corner folded down :

A note is attached to another diagram element (such as use case, actor, association) through the "Note attachment" association (broken line without arrowheads) as shown below :

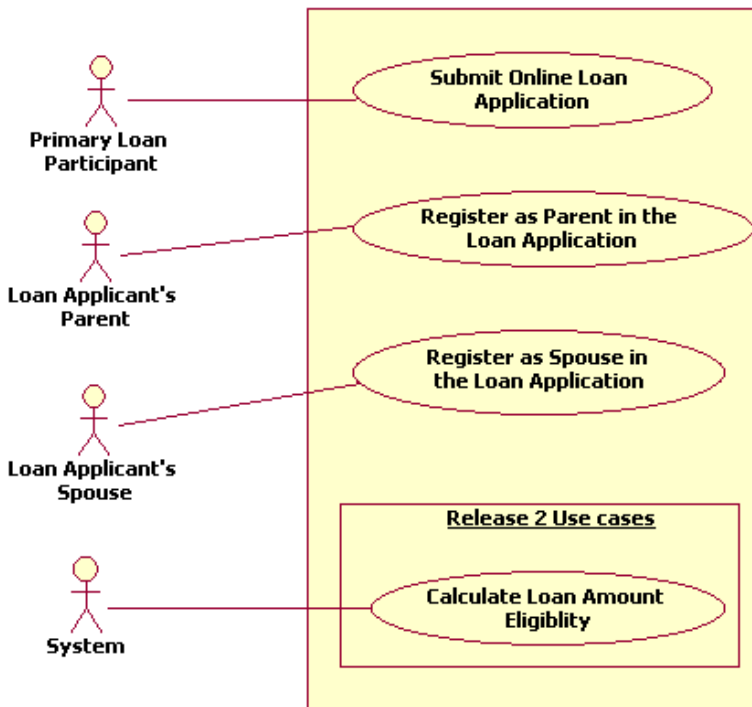


System boundary (optional)

A system boundary is a rectangle that is drawn around the use cases in a use-case diagram to separate the use cases internal to a system from the actors external to it. A system boundary is an optional visual aid in the diagram; it does not add semantic meaning to the model.

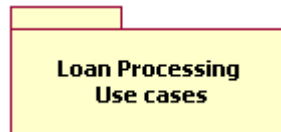


It is possible to draw multiple system boundary boxes (box inside box) to indicate release management scope - one box for a set of use cases in each release of system development, as shown in the example below.



Packages

Packages are UML constructs that facilitate to organize use cases into groups to prevent a large use case diagram becoming unwieldy due to large number of use cases. They are used to organize a large diagram into several smaller ones.



Packages are depicted as file folders :

Naming standards

Actor Naming

- An actor should have a generic name that accurately reflects its role within the use case model.
- Do not use positions (such as DBA, Business Analyst) as actor names unless the position itself reflects the generic name for the actor.
- Actors should be named as singular nouns such as *Student*, *Teacher*.
- For names with more than one word such as "*Loan Applicant*", each word should begin with upper case and words should be separated by a space. Prepositions if any (such as "in", "to", "from") should be completely in lower case.
- The name of the actor should be displayed directly below the Actor icon.
- Do not label actor types (stakeholder, primary or secondary) for the actors on the use case diagram. Rather use the "Actor Profile" section of "*System Use case Specification Template*" to document actor types.

Use Case Naming

- Use case name should begin with a strong verb such as "Register", "Submit" etc.
- Use case names should not begin with weak verbs such as "process", "perform" etc.
- Use Case should be named using Application or business domain terminology. IT jargon should not be used.
- For names with more than one word such as "*Fill-in Loan Application*", each word should begin with upper case and words should be separated by a space. Prepositions if any (such as "in", "to", "from") should be completely in lower case.
- The name of the Use case should be displayed at the center of the Use case icon.

Association Naming

- Do not attach any custom labels for associations. Follow the conventions mentioned below which are supported by most UML tools :
 - "Actor-Use case" association : solid line, no arrowheads, no labels (—).
 - "Actor-Actor" association : solid line, closed arrowhead, no labels (→).
 - "Use case-Use case extend" association : broken line, open arrowhead, labeled as "«extend»" (- - - - - $\langle\langle\text{extend}\rangle\rangle$).
 - "Use case-Use case include" association : broken line, open arrowhead, labeled as "«include»" (- - - - - $\langle\langle\text{include}\rangle\rangle$).
 - "Use case-Use case" inherits association : solid line, closed arrowhead, no labels (→).
- Most UML tools assign default names to associations. Do not modify these default names.

Package Naming

- Package name should be short comprising of few words.
- Try to follow same naming convention as for Use cases.
- Ensure that package name correctly reflects the elements it is grouping.

Modeling standards

Actor modeling

- Ensure that different actor names are not used to describe the same role duplicating the actor.
- It is recommended to place the Primary actor(s) at the top left corner of the use case diagram for attention and readability.
- It is recommended to draw Actors outside the use case diagram (and outside the system boundary box, if exists) rather than in the middle of the diagram.
- Every actor should be involved with at least one use case.
- A single actor may use (or trigger) more than one use case.
- Do not draw association between actors. Association should be drawn between actor(s) and use case(s) only. An exception to this is *Actor Generalization*, which was explained earlier in "Associations" section.
- Use Actor generalization sparingly and avoid creating of large hierarchy of Actor generalizations.

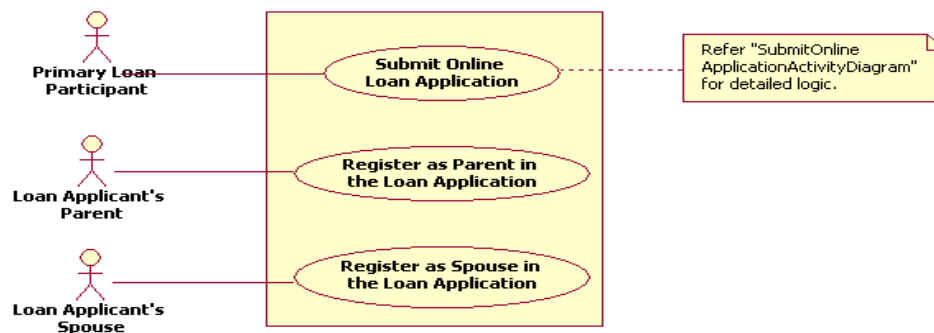
Use Case modeling

- Always draw the Use case as the horizontal ellipse figure and do not use stereotypes.

- Sequence of execution of use cases is not important. But if required, this can be reflected in the diagram by way of stacking the use case icons in the preferred order of execution.
- Place the most important use cases at the top left corner of the use case diagram for attention and readability.
- Always draw use cases inside the system boundary box (if one exists) and never outside.
- Avoid functional decomposition of use cases.
- Do not draw association between use cases. Association should be drawn between actor(s) and use case(s) only. An exception to this is *Use case reuse*, which was explained earlier in "Associations" section.
- Every use case should be associated with at least one actor.
- A single use case may be associated (or triggered by) more than one actor.
- If <<extend>> association is used between two use cases, then place the extending use case a level lower than the base use case in the use case diagram as shown in the example below :



- If an individual use case has an Activity Diagram associated with it, then mention this on the use case diagram as a *Note* text as shown in the example below :



Association modeling

- Do not miss out genuinely existing association between actors and use cases.
- Use <<include>> only when it is worthwhile the encapsulation of a use case service for later reuse or for separating a concern in the requirements management.
- *As a thumb rule, if there is a part of a base use case that represents a functionality of which the use case only depends on the result (not the method used to produce the result), then factor that part out as a different use case (included use case) and invoke it in the base use case using <<include>> association.*
- There are two most common occasions for using the <<extend>> association and creating extension use cases :
 - When there is a need to add new requirements to the frozen requirements without disturbing the base use cases.

- When there is a need to activate asynchronous services to interrupt the base use case under certain conditions.
- *As a thumb rule, if there is a part of a base use case that is optional, or not necessary to understand the primary purpose of the use case, then factor that part out as a different use case (extended use case) in order to simplify the structure of the base use case. The extended use case is implicitly inserted in the base use case, using the <<extend>> relationship.*
- While using <<extend>> association, ensure that the base use case is complete and executable regardless of the extending use case.
- While using <<extend>> association, ensure that the base use case is frozen.
- While using <<extend>> association, ensure that the base use case is clearly defining the extension points and extending conditions.
- Always document the use case extending conditions on the use case diagram (using "Note" element) as well as inside Use case specifications.
- Do not show extension points in the Use case icons to avoid diagram cluttering.
- Always describe Extension points in text form rather than only using labels or short names.
- Avoid deeply nested <<include>> relationships leading to function decomposition.
- While using <<include>> association always mention the inclusion point in the "Use case specification" document.
- *As a thumb rule, if there are use cases that have commonalties in behavior and structure and similarities in purpose, then factor out their common parts into the base use case and inherit it in the child use case using "inherit" association. The child use cases can insert new behavior and modify existing behavior in the structure they inherit from the parent use case.*

Note modeling

- The *Note* element has no name.
- Use notes judiciously so as not to clutter the diagram.

System boundary modeling (optional)

- System boundary boxes are optional – so avoid using them if it wouldn't add communication value to the diagram.
- Use System boundary box only if the model contains large number of use cases and/or you want to indicate release management priorities, if any.

Package modeling (optional)

- Packages are optional – so avoid using them if it wouldn't add value to organizing the use cases in the use case diagram.

- Use packages to group or organize use cases. Do not use packages to group just the Actors alone without use cases since it does not serve any purpose.
- Use packages in a cohesive manner.
- Group only related use cases into the package and do not group unrelated use cases.
- As a thumb rule, a single package should contain no more than 7-10 elements (use cases and actors).
- The "included" and "extending" use cases should be grouped into the same package as their base (parent) use cases.

Specification standards

The following sections describe the standards to be followed during the Use case specifications (documenting use case in text form). Instructions on what to fill in the various sections of the specifications are available within the template.

- Always use the standard template "*System Use case Specification Template.doc*" for documenting the use case in narrative form.
- Each use case must have its own separate specification and do not combine multiple use cases into a single specification.
- Ensure that use case specification text (element names, terminology etc) is consistent with the use case diagram.
- The Use case ID should be in the following format : Application acronym-SU###, where SU stands for "System Use case". Example : ILA-SU012 indicates the System use case 12 of the ILA application.
- In the *Flow of events* section, describe only the events that belong to the use case, and not what happens in other use cases or outside the system.
- Ensure to document the *non-functional requirements* on the use case that will influence the system design (but are not reflected elsewhere in the diagram).
- Use *Pre- and Post-condition* sections only if it adds value to the use case. Ensure that Pre- and Pos-conditions are applicable to the whole use case and not to any specific flow of event. While describing the Pre-condition, ensure that the pre-condition is a constraint on "when a use case can start" and not the event that triggers the use case. While using post-conditions together with <<extend>> association, ensure that the extending use case does not introduce a sub-flow that violates the post-condition in the base use case.
- Using named *extension points* helps separate the specification of the behavior of the extending use case from the internal details of the base use case. The base use case can be modified as long as the names of the extension points remain the same and it will not affect the extending use case. Ensure that the text describing the flow of events of the base use case is not loaded with details of where behavior might be extended into it.