Sample Bank ATM
Use-Case Modeling Guidelines

Version 1.0

Table of Contents

1. Document Control Information

1.1 Revision History

Date	Version	Description	Author

1.2 Reviewers and Approvers

Reviewer	Functional Area

2. Introduction

2.1 Purpose

This document describes use-case modeling guidelines, and may aid teams that are new to using a use-case driven approach to requirements management. It is simply a guide. The goal of these guidelines is to provide project standards for the development and maintenance of requirements with use cases, and to ensure a consistent look and feel of all use-case related artifacts.

2.2 References

- *Use Case Modeling*, Kurt Bittner, Ian Spence. June 2003.
- *The Unified Modeling Language References Manual, Second Edition*, James Rumbaugh, Ivar Jacobson, Grady Booch, July 2004.
- Rational Unified Process
- OMG Unified Modeling Language Specification, Version 2.0. (http://www.omg.org)

2.3 UML Terms

2.3.1 Actor

An actor represents a role that a human, hardware device, or another system can play in relation to the system. It is external to the system. It imposes requirements on what the system must do.

The actor's name must clearly denote the actor's role. Describe an actor by writing a brief description that includes the actor's area of responsibility and what the actor needs the system to do.

2.3.2 Use Case

A use case is the specification of a set of actions performed by a system, which yields an observable result that is, typically, of value to one or more actors or other stakeholders of the system. (UML2.0)

2.3.3 Use-Case Model

A use-case model describes a system's functional requirements in terms of use cases.

2.3.4 Use-Case Diagram

A use-case diagram is a UML diagram that shows the relationships among actors and use cases within a system.

2.3.5 Communicates Association

The relationship between an actor and use case is illustrated by an association symbol – a solid path between the use case and the actor symbols. (*The Unified Modeling Language Reference Manual*, Second Edition.) This solid path is referred to as a communicates association. A communicates association between an actor and a use case indicates that they interact, that is, the actor participates in and communicates with the system containing the use case. Communicates associations **do not describe data flow**. Interactions can be mechanical, electrical, data, audible, visual, or any combination of these. For example, an actor may press a button in the system (mechanical stimulus) and the system could turn on a light (visual response).

A communicates-association is represented in UML diagrams as a solid line. The line represents a two-way or whole dialog between the actor and the system.

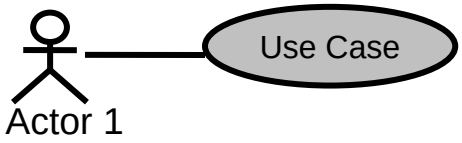

Figure 1 Communicates Association

3. How to Describe a Use Case

3.1 General Style

The use cases will be written using the RUP use-case specification template.

3.2 Use Case Name

The use-case name will be unique, intuitive, and explanatory so that it clearly and unambiguously defines the observable result of value gained from the use case.

Each use-case name will describe the behavior the use case supports. The name will combine both the action being performed and the key element being "actioned". Most often, the name will be comprised of a simple verb – noun combination. The use case should be named from the perspective of the actor that triggers the use case.

A good check for the use-case name is to survey whether customers, business representatives, analysts, and developers all understand the names and descriptions of the use cases, and that the observable result of value from the actors perspective is defined.

3.3 Use Case Brief Description

The brief description of the use case should reflect its purpose. As you write the description, refer to the actors involved in the use case, the glossary, and, if you need to, define new concepts.

3.4 Use Case Flows of Events Style

The use-case flows will make use of labels to describe the steps at a high-level, followed by descriptions of what the actor does and what the system does under the label. These labeled steps will be referenced in alternative flows and subflows.

Labels will be in a bold font, underlined, and can be formatted in Microsoft Word as headings. Sub-steps (use case detail) will be described within each labeled step.

3.5 Basic Flow

The first step of the use case description will start with "The use case begins when the actor…"

Each time the interaction between the actor and the system changes focus (between the actor and the system), the next segment of behavior will start with a new paragraph. Begin first with an actor and then the system.

When possible, each step in the flow of events should show a roundtrip of events, from actor to system, and from system to actor, usually in the form of messages:

- What the actor does: <Actor> messages <System> , and
- What the system does in response: <System> messages <An Actor> ...

Making each step a roundtrip:

- Ensures testability
- Ensures adherence to the purpose of a use case
- Makes sequence diagrams far easier to develop and read.

The last step of the use case description will end with "The use case ends."

3.6 Alternative Flows

Alternative flows will be used to document regular variants, odd cases, and exceptional (error) flows.

Each alternative flow will explicitly and clearly define all of the possible entry points into the flow, and will conclude with all of the possible exit points from the flow.

The alternative flow will also state explicitly the exit point and where the actor continues to next, that is, whether the flow returns to a specific step in the basic flow, or ends.

Alternative flows are described in a separate section. The basic flow will not reference any alternative flows.

3.6.1 Using Alternative flows

A specific alternative flow occurs at a specific step in the basic (or other) flow. A generic alternative flow can occur anywhere in another flow. For specific alternative flows, the following information is detailed:

- The start location in the basic or another sub-flow where the alternative flow is triggered.
- The condition that triggers its start.
- The actions taken in the alternative flow.
- Where the basic or another sub-flow is resumed after the alternative flow ends.

If the use case ends in the alternative flow, state that "The use case ends" in the alternative flow.

For generic alternative flows, there is no start location because a generic alternative flow can begin anywhere.

3.7 Subflows

Where the flow of events becomes cluttered due to complex behavior, or where a single flow exceeds a physical printed page in length, subflows can be used to improve clarity and manage complexity. Subflows can also facilitate the reuse of flows within the same use case. Subflows will be written by moving a self-contained, logical group of detailed behavior to a subflow, and then referencing this behavior in summary form within the flow of events. A subflow can be thought of as an "internal include".

The difference between an alternative flow and a subflow is that alternative flows insert themselves into another flow. The flow it inserts itself into has no knowledge of the alternative flow. An alternative flow may also resume at any place within the use case. A subflow is explicitly called from a flow. When a subflow complete, it always returns to the line that follows the line from which it was called. A subflow is similar in concept to a subroutine call in some programming languages.

Subflows are not listed as part of a scenario. By definition, if the calling flow is in the scenario, the subflow is also in the scenario. As with any other flow, a subflow may have alternative flows.

3.7.1 Using Subflows

Subflows are described in their own section of the use case. Subflows have an identifier in front of them (S1, S2, and so on) to identify them as subflows.

Subflows called out in the flow of events will be identified in bold, italics text.

3.8 Preconditions and Postconditions

The use-case specification will include a set of conditions (also referred to as assumptions) that are expected to be true before the use case begins (preconditions), and after the use case has ended (postconditions). Note that the use case may end in a number of ways, and each postcondition should be described accordingly.

3.8.1 Use of Preconditions

Preconditions describe the state the system must be in before the use case can start. All preconditions must be true. Preconditions reduce the need for validation inside the use case, but cannot be used to describe things outside of the system.

3.8.2 Use of postconditions

Postconditions describe the state of the system at the end of the use case. Postconditions are guaranteed to be true at the end of the use case, regardless of how the use case ends. Any (but at least one) of the postcondition will be true.

3.9 Use of Scenarios

Scenarios represent an instance of a use case. It is one flow through a use case. Document as many scenarios as are required to understand the system being developed, but be absolutely sure to document architecturally significant and/or high-risk use case scenarios.

3.10 Use of Glossary Terms

All terms used in a use case will be defined in the project glossary. If a term exists in a use case that does not exist in the glossary, the term needs to either:

1. Be added to the glossary, including a brief description (maximum of one paragraph).

2. Be changed in the use case to reflect the correct term defined in the glossary.

Terms that are specific only to a particular use case may be defined in the glossary section of that use case.

Items defined in the glossary will be in bold font.

3.11 Use of Interaction Diagrams

In some cases, it is beneficial to include, in addition to the textual flow of events, an interaction diagram to illustrate the high level flow of events of the use case. IBM Rational Software Architect is the recommended tool to use to draw the sequence diagram. Include only the communication between the actors and the boundary objects (covering both the input and the output messages) and treat the system as a black box. Use boundary objects with logical names as defined in the use case flow of events, without assigning them to classes at this point.

Interaction diagrams are optional.

3.12 Use of Activity Diagrams

In some cases, an activity diagram can add value by helping to define, clarify, and complete the flow of events in the use case. IBM Rational Software Architect can be used to create activity diagrams. A good practice is to create activity diagrams for complex use-cases (containing several alternative or exceptional flows). The activity diagram shows a decision tree of the flows in the use case.

Activity diagrams are optional.

3.13 Additional Guidelines

3.13.1 Inline conditional and repetitive behavior

Using conditional and looping statements such as IF, THEN, ELSE, WHILE, REPEAT, and UNTIL in the flows of events makes it difficult to identify scenarios. Therefore, they are prohibited. All conditional and repetitive behavior must be expressed in alternative flows (see *Figure 2 Alternative Flow to Describe Repetitive Behavior*). Activity diagrams can be used to visualize alternative flows.

The following statements are permitted in alternative flows:

- IF – used to express the condition the alternative flow is executed
- RESUME – used to express where the alternative flow resumes in the use case.

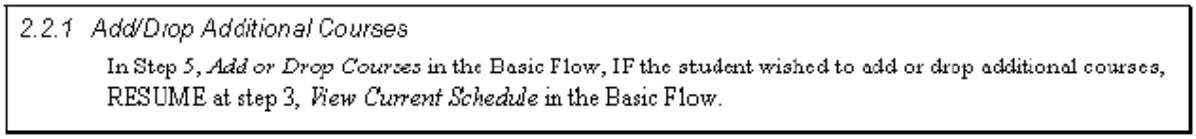

Figure 2 Alternative Flow to Describe Repetitive Behavior

3.13.2 Consistent Use of Actor Name(s)

The use-case specification will be written using consistent actor name(s). Do not refer generically to “the actor”; instead, use the actual name used to uniquely identify or define the actor. Ensure that actor naming is clear and unambiguous.

3.13.3 Consistent use of the imperative: Will

System requirements within the use cases will be written using the imperative. The term “Will” has been chosen in favor of “Shall” or “Must” to describe requirements consistently. Avoid the use of passive terms that imply the requirement is optional or undefined such as “should”, “possibly”, “etc”, “might” or “may”.

3.13.4 Use of “Action” Terms

3.13.4.1 Define where the system is responsible for presenting the Action Option

The use case will explicitly state where the system is responsible for presenting an action as an available option for the actor to select. In most cases, the available options should be presented as part of the basic flow, and should be referenced as the entry point in the first statement in the corresponding alternative flow.

3.13.4.2 Consistent use of the term throughout the Use Case

The use of terms such as New, Modify, Cancel, Delete, OK, and Print will be consistent throughout the use case: The same logical action will not be referred to using different terminology. Ensure that the Action Terms used in the Alternative Flows match those used in the basic flow.

3.13.5 Use of placeholders for missing detail (TBD or None)

Where information is not yet defined or not yet decided, the use case will include a reference to the issue or element and will include the placeholder "TBD". If the section will be blank, use the placeholder "None".

3.13.6 Definition of and Reference to Supplementary Specifications

Additional requirements that cannot be described naturally during the flow of events will be defined as supplementary requirements. Those that are specific to a use case will be defined in the Special Requirements section of the use-case specification.

Those requirements that are applicable system-wide, especially those of a non-functional nature, will be defined in the separate supplementary specification document.

3.13.7 User Interface

The user interface will not be specified in the use case.

See *Figure 3 Describing the User Interface* for examples.

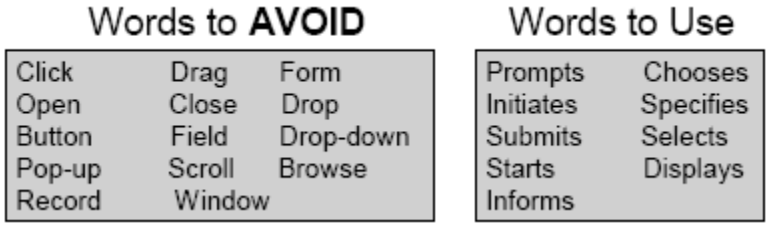

Figure 3 Describing the User Interface

3.13.7.1 Crosscheck with UI Prototype/ Design

The use case contents will be cross-checked against the UI Prototype/ Design to ensure that no system requirements are missing from the use case or the UI Prototype/ Design.

4. Model Structuring

*[This information on structuring use-case model using generalization, include, and extend relationships is not covered in the **Writing Good Use Cases** course. Structuring the use-case model using these relationships should be performed by architects after use cases are detailed. It is not essential to writing good use cases.]*

4.1 Overview

To structure the use-case model, we have three kinds of relationships, <<include>>, <<extend>> and generalization relationship.

4.2 <<include>> and <<extend>> relationships

<<include>> and <<extend>> relationships can be used to:

1. Factor out behavior that is in common for two or more use cases.
2. Factor out behavior from the base use case that is not required to understand of the primary purpose of the use case, only the result of it is important.
3. To show that there may be a set of behavior segments that may be inserted at an extension point in a base use case.

Avoid using <<include>> and << extend>> relationships because they tend to clutter and confuse rather than simplify the use-case model. The best practice is to avoid this type of decomposition initially. Consider using these

relationships later in the process and only when common behaviors can be found in multiple use cases. They should only be used where they add value by helping to simplify and manage the use-case model.

4.3 Use-Case Generalization

If there are use cases that have commonalties in behavior and structure and similarities in purpose, their common parts can be factored out into a base use case (parent) that is inherited by addition use cases (children). The child use cases can insert new behavior and modify existing behavior in the structure they inherit from the parent use case.

4.4 Actor Generalization

Actor generalization is used to simplify the use-case diagram and avoid the "chopstick effect" of communicates associations when multiple actors all execute the use case for the same purpose, that is, when multiple (i.e. actors playing the same role.) In general, actor generalization can be used to better define the different roles played by the users of the system to be developed. This is useful in applications with different "categories" of end users. In this way, only relevant functionality will be presented to each category of users, and you can control the access rights based on this grouping.

In the following example, the three actors, doctor, nurse, and aide, all perform the use case *Read Chart*. Rather than have all three actors, each with a communicates association to the Read Chart use case, you can identify an *abstract actor* and use actor generalization between the actors. The following example shows an *abstract actor* Medical Worker (no one is employed as a medical worker) that represents the role that the three concrete actors play when reading the medical chart. This actor generalization simplifies the use case diagram.

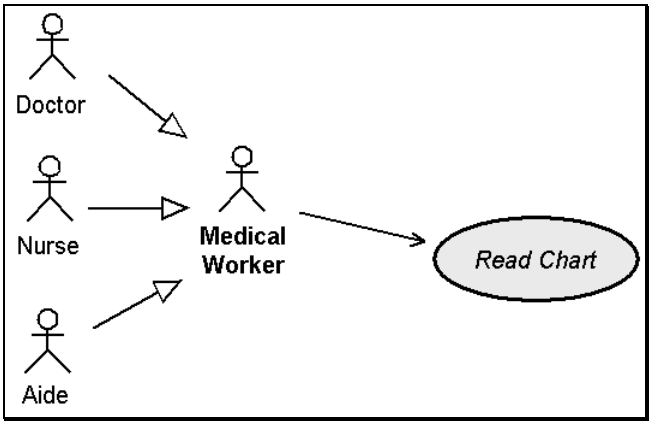

Figure 4 Actor Generalization