# *A Use Case Template:*
## *draft for discussion*

Derek Coleman
Hewlett-Packard Software Initiative
derek_coleman@hp.com

## *1. Introduction*

One of the more surprising things about the UML standard is the lack of detail on the structure of use cases. The UML semantics [1] state that "A use case can be described in plain text, using operations, in activity diagrams, by a state-machine, or by other behavior description techniques, such as pre-and post conditions. The interaction between the use case and the actors can also be presented in collaboration diagrams." However no format is presented for any of these alternatives.

In our experience, most users find it convenient to follow Jacobson's original style [2] of natural language descriptions embedded in a table. The problem is that the original proposal was incomplete. The absence of a UML standard means that users are likely to continue to invent their own format and thus continue the proliferation of use case variants. Apart from the unnecessary duplication of effort the lack of a common format impedes the growth of understanding of what makes a good use case. This document outlines a syntax and informal semantics for use case templates and for the uses and extends relationships between use cases.

## *2. Use Case Template*

The template, shown in table 1, has eight fields. The left-hand column shows the fields and whether they are optional. The right hand column briefly describes the purpose of the field.

| | |
|---|---|
| **Use Case** | *Use case identifier and reference number and modification history* |
| **Description** | *Goal to be achieved by use case and sources for requirement* |
| **Actors** | *List of actors involved in use case* |
| **Assumptions** | *Conditions that must be true for use case to terminate successfully* |
| **Steps** | *Interactions between actors and system that are necessary to achieve goal* |
| **Variations** *(optional)* | *Any variations in the steps of a use case* |
| **Non-Functional** *(optional)* | *List of non-functional requirements that the use case must meet.* |
| **Issues** | *List of issues that remain to be resolved* |

**Table 1 : use case template**

The following sections discuss the form and semantics of each field.

### *Use Case*

Each use case should have a unique name suggesting its purpose. The name should express what happens when the use case is performed. It is recommended that the name be an active phrase, e.g. "Repairing_Cellular_Network". It is convenient to include a reference number to indicate how it relates to other use cases. The name field should also contain the creation and modification history of the template preceded by the keyword **history**.

### Description

Each use case should have a description that describes the main business goals of the use case. The description should list the sources for the requirement, preceded by the keyword **sources**.

### Actors

Lists the actors involved in the use case. Optionally, an actor may be indicated as primary or secondary. A *primary* actor is one having a goal requiring the assistance of the system. A *secondary* actor is one from which the system needs assistance to satisfy its goal. Implicitly the system under discussion is also an actor, although it is not listed.

### Assumptions

Lists all the assumptions necessary for the goal of the use case to be achieved successfully. Each assumption should be stated as in a declarative manner, as a statement that evaluates to true or false. If an assumption is false then it is unspecified what the use case will do. The fewer assumptions that a use case has then the more robust it is. Use case extensions can be used to specify behavior when an assumption is false.

### Steps

The sequence of interactions necessary to successfully meet the goal. The interactions between the system and actors are structured into one or more steps which are expressed in natural language. A step completes when all its component interactions have completed. A step has the form

<sequence number><interaction>

If there are multiple steps, then each step must have an integer sequence number showing its position in the list of steps. Steps are initiated in order, in accordance with their sequence number. The default assumption is that each step is completed before the next is started, e.g. in

```
1. interaction a
2. interaction b
```

*interaction a* starts and completes before *interaction b* starts.

The execution of one or more steps may be made conditional by using a programming language-like if-then-else constructs, e.g. in

```
1.  IF test THEN 1.1 interaction a
                 1.2 interaction b
            ELSE 1.3 interaction b
```

either *interaction a* and *b* happen or *interaction c happens* depending on whether *test* is true. Note that Dewey decimal numbering is used to indicate sub-steps.

Similarly programming language iterative constructs may be used to indicate the repeated execution of a group of interactions, e.g. in

```
1. REPEAT
       1.1. interaction a
       1.2. interaction b
   UNTIL no more
```

*interaction a* and *b* happen repeatedly until *no more* is true.

**Concurrency in Steps**
A number of interactions can be specified to happen concurrently *within* a step[1] by using keywords such as IN PARALLEL <interaction> || <interaction> || … or CONCURRENTLY DO <interaction> || … A step containing concurrent interactions completes when all its interactions have completed, e.g. in

*1.* IN PARALLEL *interaction a* || *interaction b* || *interaction c*
*2. interaction d*

*interactions a*, *b* and *c* operate concurrently. The order in which they start and complete is unspecified. Step 2 does not start until they have all completed.

Alternatively a step may consist of a number of parallel sub steps in which case, the interactions are given sequence numbers. In this case the sub steps start sequentially but their execution can overlap e.g. in

*1* . IN PARALLEL *1.1 interaction a* || *1.2 interaction b* || *1.3 interaction c*
*2. interaction d*

*interactions a*, *b* and *c* start in order but the order in which they complete is unspecified.

**Referring to steps and interactions**

The sequence numbers may be used to reference steps from elsewhere within the use case or an extension of the use case. References to steps are indicated by prefixing a # character. Thus #1.2 is a reference to step 1.2.

If it is necessary, individual interactions within a step may be labeled by including an alphanumeric string followed by a colon, e.g. in

*1. interaction a* || *X23: interaction b* || *interaction c*

*X23* is the label of *interaction b*.

## *Variations*

Further detail about a step may be given by listing any variations on the manner or mode in which it may happen.

<step reference> < list of variations separated by **or**>

For example, if the original step is

1. Buyer calls in with a purchase request.

Then the variations might be

**#1.** Buyer may phone in, **or**
fax in, **or**
use web order form.

## *Non Functional Requirements*

The nonfunctional requirements are listed in the form:
<**keyword**> : < requirement>

Non-functional keywords include, but are not limited to **Performance**, **Reliability, Fault Tolerance**, **Frequency,** and **Priority**. Each requirement is expressed in natural language or an appropriate formalism.

## *Issues*

List of issues awaiting resolution. There may also be some notes on possible implementation strategies or impact on other use cases.

[1] Often it is sufficient to express any concurrent behavior in a use case through the use of natural language phrases such as “at the same time as”.

## 3. Example of a Use Case

The example below shows a use case that describes the interactions involved in repairing a cellular network. The business goal of the (primary) actor the Operator is to repair the network. The other active participants are the cellular network itself and a field maintenance engineer. The source information for the use case comes from two sources viz. Operating Manual [1993] and [Jones 1998]. In order for the use case to succeed it is assumed that any changes that are sent to the cellular network will not fail. That is the network cells will change as directed.

| | |
|---|---|
| **Use Case** | 2. Repairing_Cellular_Network<br>**history** created 1/5/98 Derek Coleman, modified 5/5/98. |
| **Description** | Operator rectifies a report by changing parameters of a cell.<br>**sources** [Operating Manual 1993], [Jones 1998]. |
| **Assumptions** | Changes to network are always successful when applied to a network. |
| **Actors** | Operator (primary)<br>Cellular network<br>Field maintenance engineer |
| **Steps** | 1. Operator notified of network problem.<br>2. Operator starts repair session.<br>3. REPEAT<br>    3.1.Operator runs network diagnosis application.<br>    3.2 Operator identifies cells to be changed and their new parameter values.<br>    3.3 IN PARALLEL<br>        3.3.1 Maintenance engineer tests network cells \|\|<br>        3.3.2 Maintenance engineer sends fault reports.<br>  UNTIL no more reports of problems<br>4. Operator closes repair session. |
| **Variations** | **#1.** System may detect fault and notify operator **or**<br>Field maintenance engineer may report fault to Operator. |
| **Non-Functional** | **Performance Mean:** time to repair network fault must be less than 3 hours.<br>**Fault Tolerance:** A repair session must be able to tolerate failure of Operator's console. |
| **Issues** | What are the modes of communication between field maintenance engineer and operator? |

**Table 2 : Example Use Case**

The use case is triggered by an Operator being notified that there is a fault. There are two variations on how this may happen, either via the system or via a maintenance engineer. The use case repeats the repair process until the fault has been rectified. The field service engineer concurrently performs network cell tests and sending fault reports.

Repairing the network must be accomplished within three hours on average. Further more if the Operator's console crashes then the repair session should be able to continue. The use case description does not specify the allowable modes of communication between the operator and field service engineers, will need to be resolved.

## 4. Using a Use Case

In UML, commonalties between use cases are expressed with the **uses** relationship. The relationship means that the sequence of behavior described in a *used* (or *sub*) use case is included in the sequence of the *using* use case. Using a use case is thus analogous to the notion of calling a subroutine. Sub use cases are full use cases in their own right, and therefore can be expressed using the use case template.

Using a sub- use case in a step is expressed by a keyword such as PERFORM (or USING). For example, if Tune_Cell were a use case it could be used by the following interaction

PERFORM Tune_Cell

In allocating reference numbers to use cases it is maybe convenient to use a Dewey Decimal numbering scheme to convey the using hierarchy. Thus if use case W, with reference number n, uses three sub uses cases X, Y and Z, then X may be numbered n.1, Y n.2 etc. According to this numbering scheme, sub use cases will be numbered according to where they are first used; this is not restrictive as they may be used from anywhere solely by referencing their name.

## 5. Extending a Use Case

In UML an extension is a way of capturing a variant to a use case. Extensions are not true use cases but *change deltas* [2] that apply to an existing use case. Typically extensions are used to specify the changes in steps that occur in order to accommodate an assumption that is false.

| | |
|---|---|
| **Use Case Extension** | <extension identifier> **extends** <use case identifier> |
| **Change** | *Goal to be achieved by extension* |
| **Steps** | *Changes to use case steps.* |
| **Variations** *(optional)* | … |
| **Non-Functional** *(optional)* | … |
| **Issues** | … |

**Table 3 Use Case Extension**

The following sections describe the entries of a use case extension that differ from that of the use case template.

### *Use Case Extension*

The extension name includes a unique identifier for the extension and a reference to the use case to which the extension applies.

### *Change*

This section documents the variant of the use case in terms of the assumption that discharges.

### *Steps*

In UML the changes are expressed in terms of new or altered steps that apply to a use case at an *extension point* (i.e. a reference to a step) if some condition is true.

<step reference> **if** <condition>**then** <changes to step>

## 6. Example of a Use Case Extension

| | |
|---|---|
| **Use Case Extension** | Repair_may_fail **extends** 2. Repairing_Cellular_Networks |
| **Description** | Deals with assumption that network changes can never fail. |
| **Steps** | **#3.3. if** the changes to network fail **then** the network is rolled back to its previous state |
| **Issues** | How are failures detected? Are roll backs automatic or is Operator intervention required? |

**Table 4: Example of a use case extension**

[2] Rather confusingly UML does not make the distinction and refers to extensions as use cases.

This extension removes the assumption in the Repairing_a_Cellular_Network use case that a change to the network can never fail. The change affects step #3.3 and requires that the network be rolled back to its previous state. The use case leaves for later how change failures are detected and how the roll back is effected.

## 7. Conclusion

This document proposes a UML compatible template that can be used for documenting use cases, sub use cases and use case extensions. This is the first draft of the proposal and the reader is encouraged to send the author feedback.

The template is based on experience of the practical usage of use cases in Hewlett-Packard and elsewhere. It is hoped that the template presented here will prove simple to use because it is closer to current best practice and more complete and less ambiguous.

## 8. Acknowledgements

Thanks are due to the author’s colleagues in the Software Initiative and to the co-developers of Team Fusion. Special thanks are due to Mike Ogush, Ruth Malan, Peter Toft and Todd Cotton for their comments and input. The other major influence on the template is Alistair Cockburn’s template [3].

## 9. References

[1] UML Semantics version 1.1, Rational Software Corporation, September 1997, available at http://www.rational.com/uml/index.shtml

[2] Object-Oriented Software Engineering: A Use Case Driven Approach, I. Jacobson et al, Addison-Wesley 1992.

[3] Basic Use Case Template, Alistair Cockburn, 1996, available at http://members.aol.com/acockburn/papers/uctempla.htm