Useware Markup Language (useML)

W3C Working Group Submission 3 February 2012

Editors:
Gerrit Meixner, DFKI
Marc Seissler, DFKI

Abstract

This is a submission to the W3C Model-Based UI Working Group and describes a metamodel and XML format for defining use models.

1. Conformance

As well as sections marked as non-normative, all authoring guidelines, diagrams, examples, and notes in this specification are non-normative. Everything else in this specification is normative.

2. Introduction

This section is non-normative.

The Useware Markup Language (useML) had been developed to support the user- and task-oriented Useware-Engineering process [ZT08] (see Fig. 1) with a modeling language that could integrate, harmonize and represent the results of an initial analysis phase in one common, so-called use model in the domain of production automation and industrial environments [Mei10, MSB11].

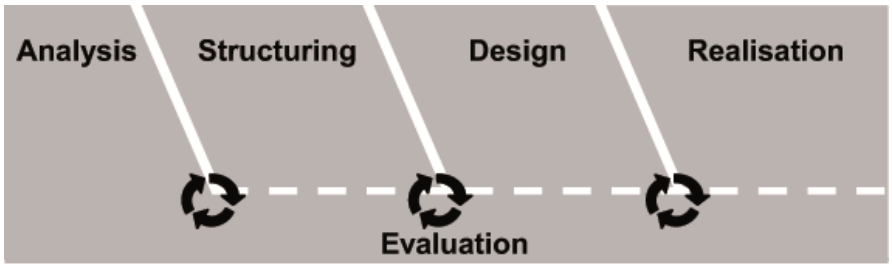

Fig 1. Useware Engineering Process

3. useML Metamodel

This section is normative.

The use model abstracts platform-independent tasks, actions, activities, and operations into use objects that make up a hierarchically ordered structure. Each element of this structure can be annotated by attributes such as eligible user groups, access rights, importance. Use objects can be further structured into other use objects or elementary use objects. Elementary use objects represent the most basic, atomic activities of a user, such as entering a value or selecting an option. Currently, five types of elementary use objects exist:

- Inform: the user gathers information from the user interface
- Trigger: starting, calling, or executing a certain function of the underlying technical device (e.g., a computer or field device)
- Select: choosing one or more items from a range of given ones
- Input: entering an absolute value, overwriting previous values
- Change: making relative changes to an existing value or item

Fig. 2 visualizes the structure of a use model.

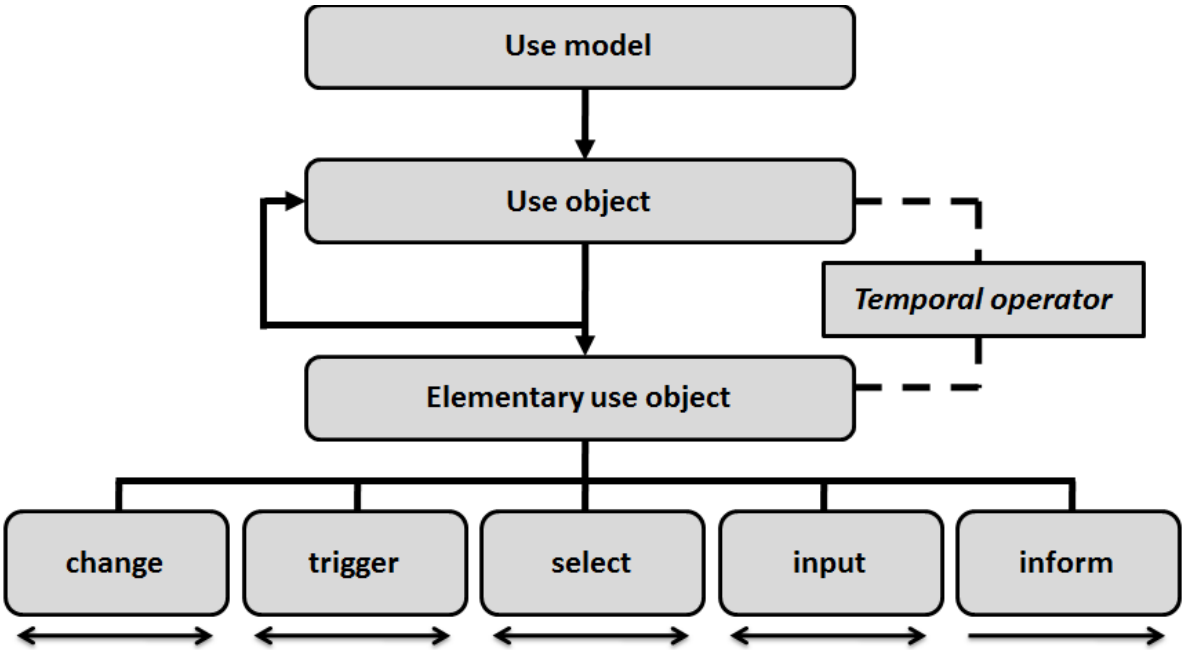

Fig. 2. Structure of a use model

The use model differentiates between interactive user tasks (performed via the user interface) and pure system tasks requiring no active intervention by the user. System tasks encapsulate tasks that are fulfilled solely by the system – which, however, does not imply that no user interface must be presented, because the user might decide, for example, to abort the system task, or request information about the status of the system. Interactive tasks usually require the user(s) to actively operate the system, but still, there can be tasks that do not have to be fulfilled or may be tackled only under certain conditions. In any case, however, interactive tasks are usually connected to system tasks and the underlying application logic. To specify that a certain task is optional, user actions can now be marked as “optional” or “required”.
Similarly, useML 2.0 can attribute cardinalities to use objects and elementary use objects. These cardinalities can specify minimum and maximum frequencies of utilization, ranging

from 0 for optional tasks up to ∞. Further, respective logical and/or temporal conditions can be specified, as well as invariants that must be fulfilled at any time during the execution (processing) of a task. UseML makes use of different temporal operators. These operators allow for putting tasks on one hierarchical level into certain explicitly temporal orders; implicitly, temporal operators applied to neighboring levels of the hierarchical structure can form highly complex, temporal expressions. In order to define the minimum number of temporal operators that allows for the broadest range of applications. The following binary temporal operators were embedded in useML:

- **Choice** (CHO): Exactly one of two tasks will be fulfilled.
- **Order Independence** (IND): The two tasks can be accomplished in any arbitrary order. However, when the first task has been performed, the second one has to wait for the first one to be finalized or aborted.
- **Concurrency** (CON): The two tasks can be accomplished in any arbitrary order, even in parallel at the same time (i.e., concurrently).
- **Deactivation** (DEA): The second task interrupts and deactivates the first task.
- **Sequence** (SEQ): The tasks must be accomplished in the given order. The second task must wait until the first one has been fulfilled.

Since the unambiguous priority of these four temporal operators is crucial for the connection of the use model with a dialog model, their priorities (i.e., their order of temporal execution) have been defined as follows:

Choice > Order Independence > Concurrency > Deactivation > Sequence

4. useML XML Scheme

This section is normative.

The XML schema for useML as a basis for an XML serialization of use models can be found in useML_schema_v2_en_smartmote.xsd.

A. References

A.1 Normative references

A.2 Informative references

[MSB11] Meixner, G.; Seissler, M.; Breiner, K.: Model-Driven Useware Engineering, In: Model-Driven Development of Advanced User Interfaces, Springer, pp. 1-26, 2011.

[Mei10] Meixner, G.: Model-based Useware Engineering, W3C Workshop on Future Standards for Model-Based User Interfaces, Rom, Italy, 2010.

[ZT08] Zuehlke, D.; Thiels, N.: Useware engineering: a methodology for the development of user-friendly interfaces, In: Library Hi Tech, Vol. 26, No. 1, 2008.