Open Issues in Industrial Use Case Modeling

Gonzalo Génova[1], Juan Llorens[1], Pierre Metz[2],
Rubén Prieto-Díaz[3], and Hernán Astudillo[4]

[1] Carlos III University of Madrid
{ggenova, llorens}@inf.uc3m.es
[2] Cork Institute of Technology, Ireland
metz@cit.ie
[3] James Madison University, VA, USA
prietorx@cisat.jmu.edu
[4] Universidad Técnica Federico Santa María, Chile
hernan@inf.utfsm.cl
http://www.ie.inf.uc3m.es/uml2004-ws6/

Abstract. Use Cases have achieved wide use as a specification tool for observable behavior of systems. However, there is still much controversy, inconsistent use, and free-flowing interpretations of use case models, in fact, not even experts widely recognized in the community agree on the semantics of concepts. Consequently, use case models are dangerously ambiguous, and there is an unnecessary divergence of practice. The purpose of the workshop was to identify and characterize some sources of ambiguity. It gathered specialists from academia and industry involved in modeling use cases to exchange ideas and proposals, with an eye to both clear definition and practical application. Some presented topics were discussed in-depth (the UML metamodel for use cases, use case instances, use cases in MDD/MDA, use case model vs. conceptual model, and tools for use cases specification), while others were left as open issues for future research. We hope our suggestions will be useful to improve the metamodel of use cases, and stimulate further research to reach a stronger coupling between the use case model and other static, behavioral and architectural models.

1 Motivation and Goals

In UML there are two main representations for use cases: *textual specifications* and *diagrams*. From a methodological standpoint, these "two worlds" have been evolving in isolation to each other. A full semantic connection between use case specification items and UML use case diagrams as initially desired by Jacobson et al. in the OOSE method does not exist. This important topic is still open to discussion and agreements, and the original impetus of the workshop comes from this dichotomy between textual vs. graphical representations for use cases.

1.1 The Graphical World

UML has been aiming to formalize use cases through object-oriented semantics by declaring the metamodel element UseCase as a subtype of Classifier, which contains

N. Jardim Nunes et al. (Eds.): UML 2004 Satellite Activities, LNCS 3297, pp. 52 – 61, 2005.

Attributes, Operations and Methods, while not defining use case documentation properties or providing a tailorable use case template. This has given use cases an explicit OO formalization as desired by Jacobson et al. in OOSE. However, the absence of a reconciling explanation of this formalization with textual use case specifications as promoted by the literature, and of guidance on how to actually document use cases, has caused a certain lack of understanding among both practitioners and researchers.

The graphical world of diagrams is dominated by use case relationships. UML's explanations of *Include* and, in particular, of *Extend* remain vague, and may even seem to be contradictory. In spite of being treated asymmetrically as two different types of use case relationships, UML's explanations do not reveal any convincing distinction; precise and unambiguous definitions of terms and semantics are missing. Moreover, UML appears to mix the instance and the type view when defining the use case relationships *Include* and *Extend*.

Another aspect where UML fails to be fully clear is regarding the meaning of use case *Generalization*: there is no indication whether subtyping and/or object-oriented inheritance semantics are meant.

1.2 The Textual World

The literature commonly emphasizes and promotes written use case specifications for functional requirements capture, which are organized according to a template; there is an implicit commitment to what a use case template should include. In contrast, use case diagrams have been used merely as an adequate graphical view on, or "entry point" to, these written specifications.

Practitioners and experts in the community frequently warn against over-emphasizing use case diagrams and strenuously advise never to neglect the use case textual specifications: in practice a use case diagram serves as a support for text but not vice versa ("a bubble does not tell us the story"). Furthermore, the techniques in the textual world are much more expressive and powerful compared to the use case relationship capabilities in UML. Finally, UML does not provide graphical modeling means for many aspects used in the textual world such as linking use cases through pre- and post-condition relations.

The current literature avoids making any commitment and prefers to highlight UML's current elusive use case relationship semantics, add to these semantics, or even arbitrarily modify these semantics, thereby keeping the practical use case concepts fuzzy. Some authors even fully discourage the use of particular use case relationships, or recommend getting rid of variety and having only a single but powerful use case relationship.

1.3 Open Areas for Research

Some of the open areas identified before the workshop are:

- Alignment of textual specification and graphical representation: use case relationships, use case standard templates, use case contracts, any information missing or extra in the two representations.

- Little semantic connection between use case specification items and UML use case diagrams. In particular, UML lacks support for the connection proposed by Jacobson et al. in OOSE.
- Collaboration vs. participation among actors of a use case. Actors may have a collaborative or participatory role in a use case, yet UML diagrams do not allow distinguishing them.
- Functional vs. structural view of use cases. Use cases may be expanded to represent functional characteristics of parts of systems, yet this expansion is not possible in UML's graphical view.
- Relationships among use cases, composition: UML allows include and extend, yet composition has a different semantics; some meaningful relationships could be borrowed from other notations, such as "precedes" from OPEN/OML and "mitigates" from Misuse Cases; dependency information encoded in pre- and post-conditions cannot be depicted graphically either.

1.4 Organization

The workshop was organized by Gonzalo Génova (Carlos III University of Madrid, Spain), Juan Llorens (Carlos III University of Madrid, Spain), Pierre Metz (Cork Institute of Technology, Ireland), Rubén Prieto-Díaz (James Madison University, VA, USA) and Hernán Astudillo (Universidad Técnica Federico Santa María, Chile).

Submitted papers were reviewed by an international team of experts composed by the organizers and Shane Sendall (University of Geneva, Switzerland), Roderick Coleman (Free consultant, Germany), Wolfgang Weber (University of Applied Sciences, Darmstadt, Germany), Sadahiro Isoda (Toyohashi University of Technology, Japan), Joaquin Miller (X-Change Technologies, USA), Guy Genilloud (IT/business consultant, Switzerland), Paul Bramble (Independent consultant, USA) and Bruce Anderson (Managing Consultant, Application Innovation, IBM Business Consulting Services, UK). Each paper received between 2 and 4 reviews before being accepted.

2 Initial Positions of the Authors

The initial two sessions of the workshop were devoted to presentation of the accepted papers, which represented a good mixture of experiences and researches both from academia and industry, as was one of the goals of the workshop. The authors came to the workshop with the following positions:

- Bruce Anderson [1]. My point of view comes from practice, from wanting UML to illuminate and support my work. While use cases can usefully be considered at various levels of formality, I would like an underlying representation that allows clear semantic relationships between use cases and other artifacts, and in particular business process models, data models, test plans, system interface objects and business rules. UML should model accurately and informatively the ways in which use case models are structured, in particular for reuse and comprehensibility. This requires the metamodel to include detail at the level of steps and alternatives.

- Nelly Bencomo, Alfredo Matteo [2]. Model Driven Software tries to focus on the modeling of systems independently of the platform, then using transformations. These models should be translated to specific platforms (for example in the field of middleware/distributed applications specific platforms are CORBA, .NET, Web Services etc).
- Clay Williams, Matthew Kaplan, Tim Klinger, and Amit Paradkar [3]. We argue that use case modeling should be done in the context of a rich conceptual model. Use cases are written in terms of this model using structured natural language. We also discuss problems that arise when trying to align this representation with the UML 2.0 metamodel, including metaclass misalignment and the lack of a representation for use case content. We close by discussing four applications of our representation: prototyping, estimation, refinement to design, and test case creation.
- Michal Smialek [4]. Use cases should have precisely defined notations which are comprehensible by various groups of people in a software development project. In order to meet these diverse views, several notations are necessary. These notations should be easily transformable and should have clear mappings to other models including the conceptual model.
- Sadahiro Isoda [5]. The current UML's use-case specification has a lot of problems and even nonsense. All these problems are due to three fundamental defects originated in OOSE. These are the illusionally "actors call use cases" conjecture, mixing-up designer's simulation with real execution and poor understanding of OO. The problems can be easily solved by recognizing anew what a use case is and then modeling it guided by plain OO technology.
- Gonzalo Génova, Juan Llorens [6]. In UML, use cases are meta-modeled as classifiers. Classifiers specify a set of instances, and use case instances are said to be concrete system-actor interactions. But it is not clear how an interaction can have classifier features such as attributes, operations and associations. Therefore, we challenge the notion that use case instances are interactions. We also propose a notion of use case (a coordinated use of system operations) that is very close to the traditional protocol, therefore concluding that use cases and protocols are not essentially different things.
- Guy Genilloud, William F. Frank [7]. It is shown that that the UML ontology is unnatural (at odds with English). As a consequence, the UML standard contains numerous sentences that confuse the picture between use cases, use case instances, and use case types. It is no surprise, therefore, that many use case practitioners do not understand the Extend relationship. The ontology of the RM-ODP, on the other hand, is more natural and more easily abided by. Using it, one would explain Extend for what it is, a relationship between specifications. Following this approach is key to reconciling the diagrammatic and textual specification techniques for use cases.
- Joaquin Miller [8]. I suggest we take an indirect approach to finding techniques to specify use cases using UML: look at use cases from the ODP viewpoint; choose ODP concepts well suited to specifying a use case; find

corresponding UML constructs; adapt the UML constructs as required. I arrive at: A particular use case of a certain system is a part of the community contract of a community of a certain type. That community is represented as a UML collaboration. I discuss how that community can be specified using UML.

3 Workshop Results

The remaining two sessions of the workshop were devoted to discussions and synthesis work, trying to reach agreement wherever it was possible. We first established a list of open issues and related them to the presented papers. Then the issues related to two or more papers were discussed in-depth: the UML metamodel for use cases [3, 5, 6, 7, 8], use case instances [5, 6, 7], use cases in MDD/MDA [1, 2, 4], use case model vs. conceptual model [3, 4], and tools for use cases specification [4, 5]. Other issues related to only one paper, or not particularly related to any of the papers, were not specifically discussed, but we mention them below. The following subsections summarize the discussions and agreements about the issues discussed in-depth.

3.1 The UML Metamodel for Use Cases

This was the main issue discussed, since it was addressed more or less directly by the majority of papers, and specifically by [3, 5, 6, 7]. We agreed that the chapter devoted to use cases and the use case metamodel in the UML2 Specification [UML2] is extremely confusing, with problems that originated in Jacobson's OOSE, were handed over to UML, and then retained so long. There are several inconsistent views and interpretations of use cases, all of them supported by the UML2, and each of them having its own difficulties. Many textual explanations in the Specification do not stick to the terminology used in the metamodel itself. This is very important for tool developers that try to be compliant with the UML2 Specification: if the metamodel is inconsistent, compliance becomes an impossible task, and incompatible interpretations of use cases lead to development of tools without interoperability.

We identified specifically the following problems:

- **Use case ontology.** The UML2 Specification introduces a dichotomy between type/specification (of a thing), on the one side, and instance (the thing itself), on the other side. For use cases, the terms are "use case" for the specification and "use case instance" for the individual. However, these terms are not consistently used, since "use case" very often means "use case instance" [7, section 2.1], leading practitioners to frequent confusions.
- **Use case features.** UseCase is a specialization of Classifier, therefore it inherits Classifier's structural and behavioral features, i.e. attributes and operations. There is not a single word in the UML2 Specification that explains the meaning of use case features. If use case instances are "interactions", being each instance a sequence of message instances exchanged between the system and the actors instances as they

communicate, then we cannot understand the meaning of use case attributes and operations: what is the sense of an interaction, a collaboration among instances, having attributes and operations? [6, section 2] We cannot think of an example of this. The UML Specification should clarify this.

- **UseCase as a specialization of BehavioredClassifier.** We do not understand why UseCase specializes BehavioredClassifier instead of Behavior [3, section 3.1.1; 6, section 2], since a use case is supposed to specify a sequence of actions, that is, a behavior. We do not understand why Behavior specializes Class either.
- **System behavior, Actor behavior, or both.** There are two inconsistent views about use cases in the UML2 Specification. On the one side, it seems a use case specifies actions performed by the subject (i.e., the system to which the use case apply); on the other side, it seems a use case specifies also actions performed by the actor when communicating with the subject. It is not clear, therefore, whether the use case specifies actor behavior or not [6, section 1]. This is of great importance for deciding the contents of a use case: should actor behavior be included in the use case specification, or not? Moreover, if actor behavior is included within the use case specification, then it has no sense saying that actors are associated with use cases, and communicate with them from the outside, as the explanations in the UML Specification and the graphical notation of use case diagrams indicate.
- **Use cases as types.** The UML Specification is misleading too when it says that a use case can be used to type one of the parts or a roles in a collaboration [6, section 1]: if the use case specifies the interaction, then it cannot specify one of the parts of the interaction at the same time.
- **Use cases vs. protocol interfaces.** UML2 has introduced the concept of a protocol interface with an associated state machine with the same purpose as use cases, namely, to specify system or subsystem usages [6]. What is the difference between use cases and protocol interfaces?

We left also some open questions that can serve as starting points for future research:

- **Generalization of use cases.** What does it mean? How are use case features inherited? (This requires clarification of the notion of use case features.) Does specialization mean subtyping too? (This requires clarification of notion of use case instance.) How is use case contents inherited (pre and post conditions, action steps, state machines…)?
- **Pre and post conditions.** What is the precise meaning of pre and post conditions? When should they be checked, at run time, or at specification time? Can pre and post conditions be used to establish sequential relationships between use cases?
- **Extend and Include relationships.** Are extensions and inclusions true use cases themselves, or are they mere fragments that deserve a different name (such as system action or activity)? Should they be visible in the use case description as separate artifacts, or are they merely configured in the model in a transparent way to the user?

- **Failures and alternatives.** How do we best express failures and choices? What terminology should be used?
- **Internal behaviors/algorithms.** How do we tie business rules to action steps? What is the relationship between use cases and system operations?
- **Log-in and log-off.** Should these be considered as separate use cases on their own with some relationship to other use cases (for example, through pre and post conditions), or should they rather be considered as mere fragments within other use cases?

3.2 Use Case Instances

Another major issue that was discussed is the improper use of the term "instantiation" to refer to the performance of actions specified in a use case [5, 6, 7]. We agreed that use case instances are not "things" that execute themselves or are executed by something else, therefore use case classifiers are not specifications of sets of "things". A behavior is not a thing, it is rather something a thing does (or something a group of things jointly do). In particular, a scenario is not an instance of a use case, and a use case does not instantiate a scenario. Rather, a use case is a complex, composite action template; a use case is a description of a behavior, and a scenario is also a description of behavior: the use case considers alternatives, exceptions, etc., while the scenario is a concrete path through the use case behavior specification.

We agreed, then, that use case instances are not "things", but we did not reach a full agreement on "instantiation" being a wrong term to refer to "the performance of actions specified in a use case". If not wrong, it seems at least to be confusing. The term "use case simulation" was proposed [5, sections 3.2 and 4.3] and received with interest, but it was not accepted by everybody.

3.3 Use Cases in MDD/MDA

The integration of use cases within Model Driven Development / Model Driven Architecture requires a better definition of use case contents [1, 2, 4], specifically a better definition of use case description of behavior through sequences of action steps [3, 4], use case pre and post conditions [7], and relationship between use case model and conceptual model [1, 3, 4].

The UML2 Specification allows for several textual and graphical representations of use case behavior, but does not provide any rules for transformations between different representations at the same level of abstraction. It does not provide either any rules for transformations of these representations to other artifacts at levels closer to implementation. Obviously, a use case must allow very informal descriptions that promote a good communication with stakeholders. However, to make them more precise, we suggest use case contents can be expressed also in semi-formal ways which should not hinder communication with stakeholders and which could be used as a refinement of those informal descriptions. Specifically, we identified several different directions to achieve this goal:

- **Use case model vs. conceptual model.** Establish a clear link between the concepts employed in use case descriptions and the vocabulary employed in the conceptual model.
- **Semi-formal structure for expressing steps.** Express action steps in simple sentences with a semi-formal structure (for example, subject-verb-object). This approach has been verified in several domains with good results [4]. However, maybe this is not feasible for all domains, an issue which deserves also further research.
- **Execution semantics for action steps.** Use action steps with clear execution semantics that allow execution or simulation of the use case behavior at an abstract level (for example, input/output statements, computations, alternatives/ exceptions checking, etc.) [3].

Other closely related issues were not discussed in-depth, and we left them as open questions for future research:

- Should we extend the metamodel to achieve this level of precision, or should we rather use the profile mechanism [4]?
- Should we have one notation for use case contents, or different ones for different problem domains or different groups of people [4]?
- Should OMG focus more on defining transformations related to the Computation Independent Model and Platform Independent Model (CIM-CIM and CIM-PIM)? Should these transformations be based on more precisely defined use case model [4]?
- Pre and post conditions: how are they related to the conceptual model and to the action steps in the description of behavior?
- How do we tie the use case model to the architecture, designs, user interfaces and code that implement it? This question requires an answer imperatively, since there is currently a lot of effort and research about MDA/MDD and it is not clear how Use Cases (and its benefits) are related to the definitions of Platform Independent Models (PIMs), Platform Specific Models (PSMs) and transformations among models.

3.4 Use Case Model Versus Conceptual Model

We also provide some suggestions to tie use cases to the vocabulary they use (in the business model), since it is good practice to keep the development of the use case model and the conceptual model in step during requirements work [3, 4]:

- The conceptual model must be consistent with the terms used for information items (the vocabulary) mentioned in the use case steps, with a clear mapping between them in both directions, maybe tool-supported, and expressed with a convenient notation.
- Different stakeholders may use different terms for the same concepts, therefore a good control of synonyms is required.

A promising field for research seems to be the construction of tools for automatic transformation and synchronization of the conceptual model with the use case model. This of course is possible when use cases are described by means of a restricted language (a subset of natural language with simple syntax that can be parsed). Would such tools be worth sacrificing informal language descriptions, or a richer, more natural, syntax?

3.5 Tools for Use Case Specification

Closely related to the MDD/MDA approach also is the need to develop tools that support the development of the use case model and other related software artifacts. We propose specifically:

- The UML metamodel should be carefully re-examined so that it allows building tools that simulate use case executions. The current metamodel says too little about use case content (e.g. steps), thus making this task difficult, even impossible [5, section 4.3].
- Tools should support transformations (that have well-defined semantics) between different use case representations and between use case representations and behaviors at the design level [3, 4]. Tools should also support synchronization with the vocabulary, as was stated in the previous section.

Tool support should not limit the possibilities to represent use cases with different notations serving particular domains. Thus an observation was made that this would necessitate that the tools have certain configuration capabilities. These capabilities would allow for defining templates for use case domain-specific notations, and what is more important – templates for transformations to other models.

3.6 Other Issues

Other issues more or less loosely related to the MDD/MDA approach were not specifically discussed, and we left them as open questions for future research:

- **Use case model and requirements.** What is the difference between the use case model and the functional requirements? Do we need separate artifacts to express them? How do we tie non-functional requirements to use cases?
- **Business processes and use cases.** How are they related to each other? How can we derive system use cases from the descriptions of business processes?
- **Level of abstraction and formality.** How many levels of abstraction are desirable in a use case model? Which level of detail is it desirable to reach? What level of formality can be reasonably attained by means of natural language?
- **Context for use cases.** How do we describe the data context for a use case? How do we relate data state in inclusions or extensions to their base use case?

- **Test cases.** How do we create test cases that get the right coverage?
- **Inspection of use cases.** What kind of rules can we give to guide inspection of use cases?
- **Metrics for use cases.** What kind of metrics can we establish for use cases?

4 Conclusions and Future Work

The summary of workshop discussions shows that our main interests lie all around two main poles: a) problems posed by deficiencies in the UML2 Specification regarding use cases, i.e. *semantics* of use cases; and b) the use case model in the context of MDD/MDA, i.e. *pragmatics* of use cases. We hope our suggestions will be useful to improve the metamodel of use cases, and stimulate further research to reach a stronger coupling between the use case model and other static, behavioral and architectural models. These two issues are closely related too, i.e. the quality of the UML Specification has a practical impact: a better definition of use case semantics (metamodel) is required to achieve a wider consensus about good practices in writing use cases and to build tools that can effectively support a use case driven development (MDD/MDA); also, the UML2 metamodel should not ignore well-established industrial practices.

Publication of workshop proceedings in the *Journal of Object Technology* is scheduled for February, 2005. New versions of the accepted papers will undergo a full review process before final publication, to achieve a higher degree of unification among them.

The workshop discussions were extremely participative and fruitful, and we hope there will be similar workshops at future UML Conferences (from now on called MoDELS Conference).

More information can be found on the workshop web site.

References

1. Bruce Anderson. “Formalism, technique and rigour in use case modelling”
2. Nelly Bencomo, Alfredo Matteo. “Traceability Management through Use Cases when Developing Distributed Object Applications”
3. Clay Williams, Matthew Kaplan, Tim Klinger, and Amit Paradkar. “Toward Engineered, Useful Use Cases”
4. Michal Smialek. “Accommodating informality with necessary precision in use case scenarios”
5. Sadahiro Isoda. “On UML2.0's Abandonment of the Actors-Call-Use-Cases Conjecture”
6. Gonzalo Génova, Juan Llorens. “The Emperor's New Use Case”
7. Guy Genilloud, William F. Frank. “Use Case Concepts from an RM-ODP Perspective”
8. Joaquin Miller. “Use Case from the ODP Viewpoint”