Use Cases in Model-Driven Software Engineering

Hernán Astudillo[1], Gonzalo Génova[2], Michał Śmiałek[3],
Juan Llorens[4], Pierre Metz[5], Rubén Prieto-Díaz[6]

[1] Universidad Técnica Federico Santa María, Chile
[2, 4] Universidad Carlos III de Madrid, Spain
[3] Warsaw University of Technology, Poland
[5] SYNSPACE, Germany
[6] James Madison University, USA
[1]hernan@inf.utfsm.cl, [2]ggenova@inf.uc3m.es,
[3]smialek@iem.pw.edu.pl, [4]llorens@inf.uc3m.es,
[5]pmetz@fbi.fh-darmstadt.de, [6]prietorx@cisat.jmu.edu

http://www.ie.inf.uc3m.es/wuscam-05/

Abstract. Use cases have achieved wide use as specification tools for systems observable behavior, but there still remains a large gap between specifying behavior and determining the software components to build or procure. WUsCaM 05 – "Workshop on Use Cases in Model-Driven Software Engineering" – brought together use case and MDSE experts from industry and academia to identify and characterize problem areas and promising approaches.

1 Motivation and goals

This workshop was the second in the series of Workshops on Use Case Modeling (WUsCaM-05). The first one took place in 2004 in conjunction with the UML'04 Conference, under the name "Open Issues in Industrial Use Case Modeling". The success of the first edition encouraged us to continue the series within the MoDELS Conference, focusing the workshop on the more specific topic of "Use Cases in Model-Driven Software Engineering", which was one of the main concerns of last year workshop discussions.

The integration of use cases within Model-Driven Software Engineering requires a better definition of use case contents, in particular description of behavior through sequences of action steps, pre- and post- conditions, and relationship between use case models and conceptual models. The UML2 specification allows for several textual and graphical representations of use cases, but does not provide any rules for transformations between different representations at the same level of abstraction. It does not provide either any rule for transformations of these representations to other artifacts at levels closer to implementation. This workshop aims to show how the resourceful application of use case models may help to bridge the "requirements gap" in current research and practice of model-driven methodologies.

1.1 Open areas for research

As a result of last year workshop discussions, we identified before the workshop a set of open areas for research, which can be grouped in two main topics: semantics of use cases, and pragmatics of use cases.

Semantics of use cases

- Semantic connection between the use case model and other software models (static and dynamic).
- Appropriateness of the UML 2.0 meta-model for supporting use case semantics and transformation into other models.
- Adding traceability information to use case models and their specification items.
- Refinement of collaboration and participation of actors in a use case.
- Precise notation (functional and structural) for use case specification items enabling model transformation.
- Clarification of relationships between use cases in the context of their transformation into other models.

Pragmatics of use cases

- Methods for use case application in a model-driven software lifecycle.
- Automatic transformations of use case model and its items into other models (analytical, architectural, design).
- Use case views and model views.
- Use cases composition.
- Tools supporting precise use case specification and transformation.
- Tools for use case verification and execution.
- Novel applications of use case models in model-driven development.

1.2 Organization

The workshop has been organized by Hernán Astudillo (Universidad Técnica Federico Santa María, Chile), Gonzalo Génova (Universidad Carlos III de Madrid, Spain), Michał Śmiałek (Warsaw University of Technology, Poland), Juan Llorens (Universidad Carlos III de Madrid, Spain), Pierre Metz (SYNSPACE, Germany), and Rubén Prieto-Díaz (James Madison University, VA, USA).

Submitted papers were reviewed by an international team of experts composed by the organizers and Bruce Anderson (IBM Business Consulting Services, UK), Guy Genilloud (Universidad Carlos III de Madrid, Spain), Sadahiro Isoda (Toyohashi University of Technology, Japan), Joaquin Miller (X-Change Technologies, USA), and Anthony Simons (University of Sheffield, UK). Each paper received between 2 and 4 reviews before being accepted.

2 Initial positions of the participants

The two initial sessions of the workshop were devoted to presentation of the accepted position 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:

- **Guy Genilloud, William F. Frank, Gonzalo Génova.** *Use Cases, Actions, and Roles.* Use Cases are widely used for specifying systems, but their semantics are unclear in ways that make it difficult to apply use cases to complex problems. The authors suggested clarifications to use case semantics so that use case modeling can be applied to relate automated systems to business processes and process specifications, particularly in situations where it's necessary to integrate multiple systems in support of a single business process. They discussed the original intentions of Ivar Jacobson and UML and found out that use case specifications, whether written in natural language or as interaction diagrams, are misleading as to what is a use case (instance). They considered then a more natural modeling technique, and established a relation between a use case, a joint action, and a role.
- **Rogardt Heldal.** *Use Cases Are more than System Operations.* Correctly written use cases can be an important artifact for describing how a software system should behave. Use cases should be informal enough to permit anyone in a software project to understand them, in particular the customer (often lacking a formal background). One consequence of adopting use cases to, for example, MDA (Model Driven Architecture) can be an increasing level of formalism, which can severely limit understanding of use cases. Also, too few guidelines for how to write use cases make them both hard to write and understand. Finding the right level of formalism was the topic of this paper. Heldal suggested a new way of writing the action steps of use cases by introducing "action blocks". The introduction of action blocks makes use cases more formal, but still understandable. In addition, action blocks support the creation of contracts for system operations. He also argued that treating system operations as use cases is a misuse of use cases —system operations and use cases should be separate artifacts. One should be able to obtain several system operations from a use case, otherwise there is no dialog (process) between actors and use cases. He believes that having a clear distinction between use cases and contracts will improve the quality of both.
- **Claudia López, Hernán Astudillo.** *Use Case- and Scenario-Based Approach to Represent NFRs and Architectural Policies.* Software architecture decisions pay primary attention to nonfunctional requirements (NFRs), yet use cases normally describe functional requirements. This article presented scenario-based descriptions of "Architectural Concerns" to satisfy NFRs and of "Architectural Policies" to represent architectural choices to address such concerns. The Azimut framework combines these modeling abstractions with "Architectural Mechanisms" to enable iterative and traceable derivation of COTS-based software architectures. An example was

shown using an inter-application communication problem, and its use in an MDA context was explored.

- **Hassan Gomaa, Erika Mir Olimpiew.** *The Role of Use Cases in Requirements and Analysis Modeling.* The authors described the role of use cases in the requirements and analysis modeling phases of a model-driven software engineering process, built on previous work by distinguishing between the black box and white box views of a system in the requirements and analysis phases. Furthermore, this paper described and related test models to the black box and white box views of a system.
- **Michał Śmiałek.** *Can Use Cases Drive Software Factories?* Contemporary software systems are becoming more and more complex with many new features reflecting the growing users needs. Software development organizations struggle with problems associated with this complexity, often caused by inability to cope with constantly changing user requirements. Supporting efforts to overcome these problems, Śmiałek proposed a method for organizing the software lifecycle around precisely defined requirements models based on use cases that can be quickly transformed into design level artifacts. The same use cases with precisely linked vocabulary notions are the means to control reuse of artifacts, promising the lifecycle to become even faster and more resourceful. With properly applied use case models practitioners can significantly improve the concept of "software factories" that newly emerges in the area of model driven software engineering.
- **Jon Whittle.** *Specifying Precise Use Cases with Use Case Charts.* Use cases are a popular method for capturing and structuring software requirements. The informality of use cases is both a blessing and a curse. It enables easy application and learning but is a barrier to automated methods for test case generation, validation or simulation. Whittle presented "use case charts", a precise way of specifying use cases that aims to retain the benefits of easy understanding but also supports automated analysis. The graphical and abstract syntax of use case charts were given, along with a sketch of their formal semantics.

Besides the previous authors, two other participants at the workshop asked for a slot to make short presentations on-the-fly, which were very much related to the ongoing discussions:

- **Pascal Roques** talked about how to derive use cases from business process activity diagrams.
- **Richard Sanders** presented research on using UML2 Collaborations to model Use Cases, showing benefits gained by Composite Structure diagrams compared to Use Case diagrams in UML2 («extend» and «include» being poorly integrated in the UML language).

3 Workshop results

The remaining two sessions of the workshop were devoted to discussions and synthesis work, trying to reach agreement wherever possible. We first established a

list of open issues and prioritized them for the discussion. The discussion was then centered around three main topics: *use case-driven development*, *some misuses of use cases* (and how to avoid them by adequate teaching), and *non-functional requirements and use cases*. We identified other interesting issues but we had not time to discuss them in-depth: business process modeling, use cases and UML, use cases and aspects, the gap between theory and practice, project size estimation (use case points have been unsuccessful), and completeness of use cases. The following subsections summarize the discussions and agreements about the three main issues.

3.1 Use case driven development

Use case-driven software development is an idea present and applied for many years in several methodologies (with UP being the most prominent). Unfortunately, use case models themselves cause many problems in applying them as true software development artifacts. These problems seem to arise from the fact that use cases are usually developed for multiple purposes: they should be well understandable by the users, enable good understanding of the system's external behavior, be coherent with the system's conceptual model, and finally, be easy to apply in the development process leading to the final software system. These highly diverse applications get reflected in a plethora of notations ranging from very informal "paragraphs of text" to perfectly formal "use case programming languages".

During the discussion we identified three topics which would allow us to resolve the ambiguity problem associated with use cases and would make the idea of use case development more efficient. These topics are:

- Differences and translation between use case representations for different purposes.
- Translation of use cases into design level artifacts.
- Reuse management on the use case level.

The first topic comes from noting that use cases could be represented by several notations at the same time. Each of these notations could be used for different purposes. The notations would include semi-formal text (subject-verb-object sentences) or graphs (activity diagrams, sequence diagrams). These notations would allow for different views of a particular use case for a specific reader (customer, developer, etc.). However, two important prerequisites need to be met. First of all, we should have an automatic transformation that would allow for instantaneous change of the view. Second, when writing use cases with any of the notations we should build a separate, independent vocabulary that would constitute a conceptual model of the specified system.

The second discussed topic derives from a statement that there still exists a significant gap between requirements and software being developed on the basis of these requirements. There is no "seamless transformation" path between requirements artifacts (like use cases) and design and implementation artifacts. This path can be significantly supported by introducing certain transformation mechanisms in accordance with the general ideas of model-driven software development. It can be argued that this transformation cannot be fully automatic. Design models need certain design decisions that can be made only by skillful architects and designers. However,

certain tools can support developers in developing a first draft of the design model and in keeping the design model constantly coherent with the requirements model and vice-versa.

The third topic, maybe the most important among all three, comes from observing very low levels of requirements-driven reuse in software development. This might be caused also by lack of satisfactory solutions in the previous two topics which are closely related. Requirements-driven reuse can be organized around precisely formulated use cases closely mapped (transformed) onto design, implementation and testing level artifacts. The discussion showed two possible approaches to use case driven reuse. One approach is associated with already widely known method of *software product lines*. With this approach, the reuse process would be organized around specifying commonality and variability of use cases. Another approach would be to treat use case models together with other, precisely related software artifacts (design, code, …) as complete cases. These cases could then be kept in libraries ready for future reuse. In this approach, all the effort associated with reuse is deferred until there arises a possibility of actual reuse.

It can be noted that all three of these topics are closely dependent on developments in model transformations. We stress the need to define precise transformations and mappings between artifacts on the requirements level (use cases, conceptual models, non-functional requirements) and from requirements to design.

3.2 Misuse of use cases

In spite of their having been around for years, we identified some common misuses of use cases among practitioners:

- *Reducing use cases to system operations.* There was general agreement that use cases and system operations should be separate artifacts at different levels of abstraction. Confusing them leads to use case models with too many use cases which are too small. A system operation is invoked by a message from an actor to the system. A use case typically contains invocations of several system operations, otherwise there is no dialog (process) between actors and use cases. Use cases are the starting point for identifying system operations, which is an important step in the design of the system.
- *Relationships between use cases.* Practitioners encounter real difficulties in distinguishing and properly applying «include» and «extend» relationships. Moreover, included and, more frequently, extending use cases are not usually full use cases, but mere fragments. Inheritance between use cases has not a better condition, since its meaning is not defined for the various existing use case representations (what parts of a textual specification are inherited, and how? what about graphical representations?). All of this often leads to vane uses of these relationships.
- *Describing business processes instead of use cases.* A use case describes an interaction between the actor(s) and the system which yields a valuable result. A use case does not describe a whole business process, it describes only that part of the business process where the system has a contribution.

The distinction is more necessary when we want to integrate several systems in a single business process.

- *Applying use cases even when they are not suitable.* Use cases are not adequate for describing the requirements of any kind of system. Use cases are good for dialog-driven systems, i.e. when the description of system-user interactions are useful to extract functional requirements. However, use cases are not adequate for event-driven systems, or for extraction of non-functional requirements.
- *Finding the appropriate level of detail.* Maybe this is one of the most difficult points about use cases. There is no universal solution, since the level of detail depends on the audience: it should not be the same for users and stakeholders, than for analysts and developers. A principal problem here is that use cases cannot be recursively decomposed into smaller use cases, which would ease the use of the same concept at different levels of abstraction. Besides, practitioners risk to confuse between *refinement* (evolving the use case description, e.g. by adding details, into a new version that still fulfills the same purpose) and *realization* (creation of new artifacts in the following software development steps that implement the services identified by use cases).

3.3 Non-functional requirements

Non-functional requirements (NFRs) are system-wide requirements that correspond to systemic properties, usually run-time (e.g. availability, reliability, security) or deployment-time (e.g. portability). The NFRs topic brought up a lovely discussion as well, which yielded several consensus ideas.

- *NFR cross-cut many use cases.* In general, NFRs do not pertain to a single individual task to be realized by a user, and cannot be specified with use cases since they aim to describe such individual tasks. We need a consistent way to indicate NFRs as related use cases (e.g. availability requirements for specific use cases).
- *Consistency among views.* There are several ways of describing requirements, and we need better correspondences between artifacts at the requirements level (use cases, conceptual models, non-functional requirements).
- *Taking NFRs from requirements into design.* NFRs are a key input to the architecture and design tasks, so they must be gathered and collected like functional requirements are. A problem is how to preserve and transmit forward such requirements through a phase (use case modeling) that has no major use for them.
- *Appropriateness of recording NFRs in use cases.* There were differing opinions regarding the convenience of describing NFRs in, or associated to, use cases. One position paper had explained two (existing) notations to record NFRs alongside use cases. Some felt that allowing this mix may pollute the “business task of value” sense of use cases, by forcing to consider and record system-wide assertions instead of focusing on the task being

described. Others argued that this may be so, but since in practice many analysis teams collect only use cases, it would be beneficial to associate NFRs to them, with the proviso that this information can be separated and recovered later for design (see above).

- *Formalism.* Whilst several more-or-less formal semantics have been proposed for use cases, they do not account for the complexity of describing NFRs as well. More formal notations for use cases-with-NFRs will be necessary to implement MDA's goal of (semi-)automated model translation.
- *Aspects.* Aspect-oriented modeling may be a good to describe NFRs in a use case context. One of the workshop papers had presented an approach based on architectural policies rather than aspects, and the participants wondered whether policies and aspects are congruent; work remains to be done in that regard. The interaction of aspects and use cases remains an interesting area, with some work already existing on Aspect-Oriented Requirements.

4 Conclusions and future work

The workshop discussions leave quite clear that use cases are the main way to specify functional requirements in modern systems development, yet some non-trivial problems remain to be addressed by the community vis-à-vis the representation of target problems, the supported processes, and the actual use by humans. These points were addressed above ("non-functional requirements and use cases", "some misuses of use cases", and "use case-driven development", respectively).

The two best papers presented at the workshop are also published in this volume along with the workshop report. To select these two papers, we had into account mainly the opinions of the reviewers, but we also considered the opinions of the other workshop participants (authors and attendants). The two chosen papers were "Specifying Precise Use Cases with Use Case Charts", by Jon Whittle, and "Use Cases, Actions, and Roles" by Guy Genilloud, William F. Frank and Gonzalo Génova.

Given the success of this scientific meeting, we hope there will be a third edition of this workshop series in the next MoDELS Conference. More information can be found at the workshop web site (http://www.ie.inf.uc3m.es/wuscam-05/).