Accommodating informality with necessary precision in use case scenarios

Michał Śmiałek

Warsaw University of Technology, IETiSIP, pl. Politechniki 1, 00-661 Warsaw, Poland and
Infovide S.A., ul. Kolejowa 5/7, Warsaw, Poland
smialek@iem.pw.edu.pl
http://www.iem.pw.edu.pl/~smialek

Abstract. Paper contains a proposition of notation for use case scenarios that accommodates the needs of different roles in software development projects. Some roles require simple and informal English sentences with references to the domain vocabulary. Other roles need a relation and mapping to user interface elements or messages flowing inside the developed system. These contradictory requirements lead to a conclusion that the use case notation should be a composition of several notations with precisely defined rules for their transformation. There are proposed four such notations based on structured text, interaction diagrams and activity diagrams. There is also presented a mapping between specific elements of these notations and a mapping to elements of the static domain and design models.

1 Introduction

If we ask software developers, what characteristics they need from requirements, they will most probably answer: precision in defining the system's scope. If we ask the users the same question, they will likely answer: we need understandability in the context of our everyday business. Unfortunately, it is a common observation that formal and thus precise requirements are hard to read and understand for the "normal people". On the other hand, requirements written in "common prose", and acceptable for the users, are usually too ambiguous for the system architects and designers. This might seem as a major discrepancy that the requirements analysts face in their everyday practice, and that eventually calls for some resolution from the methodologists.

A very good example of problems that have their source in the above discrepancy is the use case model together with the associated scenario model. Use cases introduced by Ivar Jacobson around 1992 [9], have already been defined in so many ways by different authors that it certainly leads to huge confusion about the actual notation and specification techniques. Alistair Cockburn back in 1997 has counted eighteen different definitions of use case [2]. These definitions differ in basically four dimensions: purpose, formality of contents, multiplicity of scenarios and formality of model. Depending on the definition, the use cases tend to be very formal or quite sketchy. Russel Hurlbut [8] summarizes almost 60 approaches towards use case specification.

These approaches can be classified in terms of notation into textual, graphical and dynamic. Textual formats include unstructured text narratives, structured descriptions (templates), semi-structured scripts, formal expressions and tables. Graphical formats employ structure, state, interaction and implementation diagrams based often on the UML notation. Dynamic formats are based on animations and dynamic visualizations of the use case narration flow. Other notations include storyboards and role playing.

This multitude of notations is caused mainly by imprecise definition of use cases by their inventor which was eventually not made more precise in the UML standard (including the latest version 2.0 [14]). It can be argued that this ambivalence of use cases is caused by many targets that they aim at. Use cases are often utilized as the driving elements for the whole software development process [9, 10]. The analysts write use cases to communicate their understanding of the prospective system's functionality. The users participate in formulating use cases to make sure their requirements are communicated well. The developers employ use cases to design architectures fulfilling the required functionality. User interface designers write storyboards based on use case scenarios. Testers design use case based test cases. If we browse different approaches summarized in [8] we can come to a conclusion that an "ideal" notation for use cases is impossible to reach. If the notation is to be general, it tends to be rather informal in its nature. More formal notations are designed for specific purposes, like user interface specification [4], automatic requirements verification [12] or test automation [5]. Some use case notations impose architectural or design decisions (like the usage of UML interaction diagrams).

In this position paper we argue that a single notation for use cases is not capable of accommodating all the needs imposed by a modern software development project. We thus propose a suite of closely related and transformable notations for use case scenarios. The primary notation in the suite is intended to serve as a balance between informality necessary for the users and precision aimed at all the system development roles. We also introduce secondary notations suitable for specific development purposes. All the notations have precise transformation associations that link appropriate component elements.

2 Looking for a "perfect use case"

In search for a use case notation suitable for the widest "audience" in a software development project, we should start with a suitable general definition. We shall use the definition offered by Alistair Cockburn [3] which is probably one of the most frequently referenced in the literature (and therefore quite representative). The definition starts by defining the scenario:

"Scenario. A sequence of interactions happening under certain conditions, to achieve the primary actor's goal, and having a particular result with respect to that goal. The interactions start from the triggering action and continue until the goal is delivered or abandoned, and the system completes whatever responsibilities it has with respect to the interaction."

Then, the actual definition of the use case:

"Use Case. A collection of possible scenarios between the system under discussion and external actors, characterized by the goal the primary actor has toward the system's declared responsibilities, showing how the primary actor's goal might be delivered or might fail."

The definition is general enough not to enforce any particular notation. However, it gives several important characteristics of a use case. The most important of them are: use case is a single service (with a single goal) offered by the developed system; use case is composed of several scenarios; all scenarios in a use case lead to the single goal or to failure and scenarios are sequences of interactions. We shall add to this definition also a requirement that all the use case scenarios should start with a triggering action performed by the user. According to this, it is only the actor (someone or something outside of the system) that can start scenarios, not the system itself.

The definition above is given from the point of view of a methodologist. Is sets necessary boundaries for the prospective notation. These boundaries place use cases in correct relations to other artifacts of the software development process and allow for proper organization of functional requirements in a software project. In order to design notations suitable for various project roles we still need to consider the point of view of these roles (see Fig. 1). In the following sections we will therefore try to look at use cases from several points of view. These points of view were communicated by participants of several commercial projects where use cases were used to control the development process.

User's point of view. Use case should describe a single service offered by the developed system. It should be written in common English with simple sentences. The sentences should use notions from the user's domain vocabulary. All the descriptions of the domain (notion definitions) should be avoided in the use case text (to prevent duplication) but should be easily accessible from within it. Sentences should be understood by diverse groups of users (e.g. from different departments). No special keywords or formal constructs are allowed. A single scenario should represent a single story without any alternative courses. Some users prefer graphical representation showing a graph of activities with alternative branches ("it shows the bigger picture").

Analyst's point of view. Use case should describe a single unit of functional requirements. It should have a well defined structure. Sentences in use case scenarios should be ordered and numbered (for easier reference). Terms in the sentences should be clearly referenced with appropriate vocabulary definitions. It should be possible for the analyst to use synonyms and switch between different user vocabularies when presenting scenarios to different groups of users. Entries in the vocabulary should be easily accessible for inclusion in the use case text. New terms used in text should be easy to define in the vocabulary. It should be possible to hide all the details of the user interface and of the system internals.

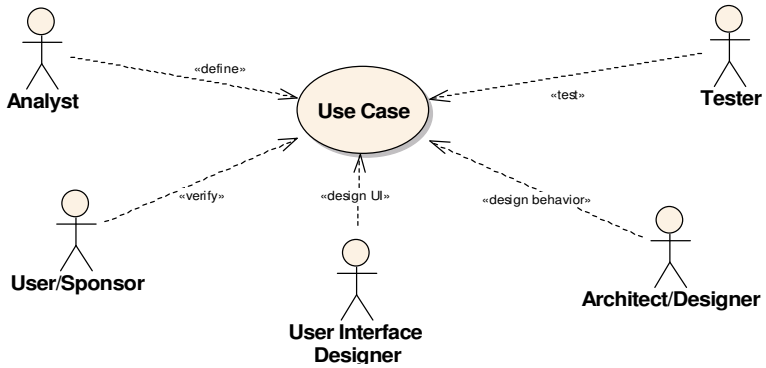

Fig. 1. Use case from different points of view

Designer's point of view. Use case should define a single unit of system's behavior to be developed. Use case should be written in a formal notation where scenarios would constitute a temporal sequence of messages (function calls etc.). The sequence should clearly reflect (a) interactions of the user with the system and (b) communication inside the system in response to the user's interactions. Messages should be connected with appropriate architectural (static) elements of the designed system including elements of the user interface. Scenarios in the form of message sequences should allow for automatic generation of code that handles these messages. Changes in the user's requirements should be easily traced to changes in message sequences, changes in the static architecture and finally – changes in code.

User interface designer's point of view. Use case should define a coherent set of storyboards that illustrate the details of the system's dialog with the user. Use case scenarios should show sequences of consecutively appearing user interface elements (screens, dialogs, menus, etc.) and interleaved user's inputs. The user interface elements should have clear links with the user interface prototype.

Tester's point of view. Use cases should be the basis for creating "test use cases". Test use cases should be associated with one or more use cases. Every test case should contain several test scenarios. Use case scenarios should be easily translatable into test scenarios. The ability of associating several scenarios with test scenarios would allow for acceptance testing where it is impossible to verify proper behavior of the system only by testing the behavior of a single use case. Note: test use cases described here have significantly different semantics than test cases as defined eg. in RUP [10].

If we consider all the requirements presented above, we come to a conclusion that a single use case should actually have several forms. For instance, the form acceptable for the users has close links with the domain model (the vocabulary). In turn, the designer's form is related to the static architectural model. One of the forms is composed of several actions, while the other is a sequence of messages. This leads us to a notation which is actually a set of distinct notations with precisely defined transformations. The idea of different forms for use cases is present in the RUP methodology [10]. Beside "use cases", RUP introduces "use case storyboards" that extend textual use

case definitions with interaction and class diagrams. However, the notation proposed in RUP in not formally defined and lacks more precise definition of transformations.

3 Use case notation metamodel with transformations

Having defined appropriate requirements for the use case notation we can now describe the notation that tries to fulfill these requirements. We will define the proposed notation by means of a MOF metamodel as it was done in [14]. Some of the elements of notation are already defined in the UML Superstructure. Names of such "predefined" metaclasses are written in bold with a reference to appropriate page in [14]. Considering the scope of this position paper we present only the most important elements of notation. The highest level elements are thus presented on metaclass diagrams. Other elements and their transformations are briefly described in text.

As declared in previous sections, the **UseCase** ([14], p. 519) shall be detailed through four distinct representations (see Fig. 2). The primary representation is structured text. Complementary to this representation is the activity graph. These representations are meant for the users and analysts. Designers use the sequence graph representation. Finally, the user interface designers use the storyboard representation. Additionally, the testers design test use cases that are not direct representations of **UseCase**. This is due to the fact that a test use case can be related to several use cases that form a testing sequence. Moreover, a single **UseCase** can participate in several test cases. Note that we leave the description of test use case notation for future work as being outside of scope of this paper.

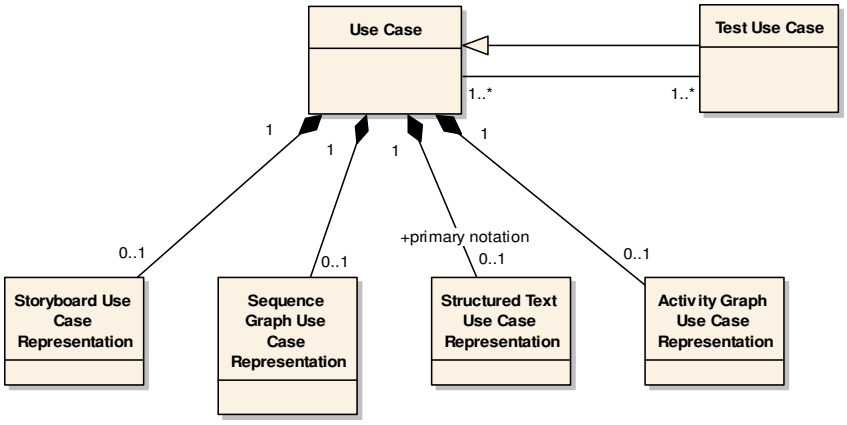

Fig. 2. General metamodel for use case representations.

The structured text representation is based on simple grammar sentences that form use case scenarios. In this grammar, sentences are composed of a subject, a verb and a direct object (SVO). In some cases, a more complex form is used, with a preposition

and an indirect object (PO). Such notation was originally proposed by Ian Graham (see e.g. [6]). Experience shows that the SVO[PO] notation is sufficient enough to represent possible interactions between users and systems. From the first sight, very simple sentences seem to limit the analysts. However, this is not the case if we relate parts of sentences to appropriate vocabularies. Simple sentences allow the analyst or the user to concentrate on interactions, while the descriptions of notions found in the interaction sequence are left out to the vocabulary. This forms a notation that is judged by the users as very clear and comprehensible. On the other hand, this notation allows for easy and unambiguous transformation to other notations.

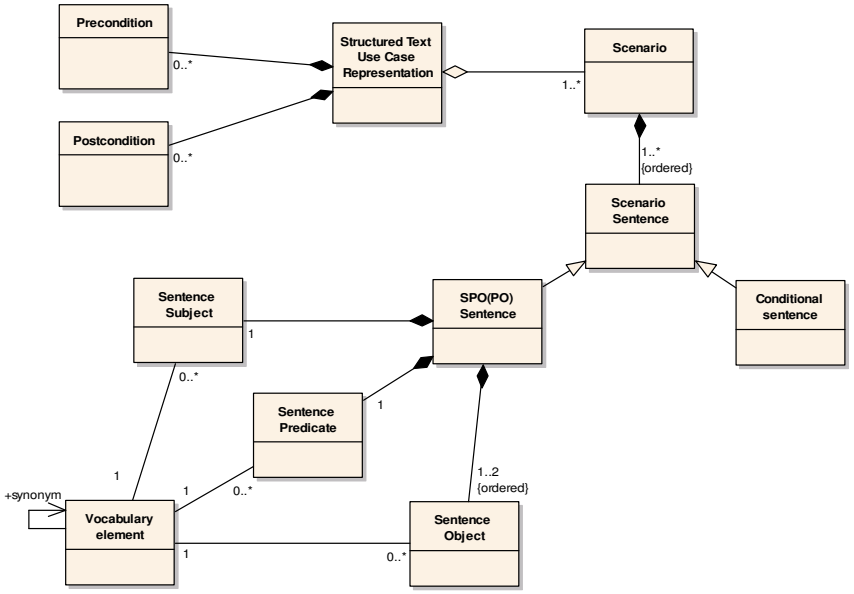

Fig. 3. Metamodel for the structured text representation.

Summary of the structured text notation is given on Fig. 3. The representation contains a list of pre- and postconditions which shall not be elaborated in more detail here, and a set of scenarios. Every scenario is a sequence of ordered sentences. These sentences are divided into SVO[PO] sentences and conditional sentences. Conditional sentences allow for synchronization between scenarios, and also allow for inclusions or extensions from other use cases (their structure is ommited here for brevity). SVO[PO] sentences form the actual course of a scenario. Every part of such a sentence (subject, verb or objects) is related to an appropriate entry in the vocabulary. Subjects relate to entries from the vocabulary of actors (instances of **Actor** – p. 512), objects relate to entries in the domain vocabulary and verbs relate to a vocabulary of predicates (operations on domain elements). These vocabularies are distinct, and appropriate constraint should be imposed on the metamodel (namely on the relations between vocabulary elements and sentence parts). Note also that on Fig. 3, the second

of the objects (i.e. the indirect object) should include a preposition which is not shown as being one of the attributes of the 'Sentence Object' metaclass).

The storyboard use case representation is very similar to the structured text representation. While the primary text notation should be UI independent, the storyboard notation allows for the inclusion of the user interface design elements. The structure of scenario sentences is very close to that shown on Fig. 3. The main difference is that sentence parts relate to other vocabularies than those in the structured case notation. These vocabularies contain descriptions (possibly graphical) of user interface elements (buttons, screens, menus etc.). Steps in the storyboard type scenarios are more fine-grained. When transforming the two models, several (possibly one) storyboard sentences map to a single structured text sentence.

Activity graph representation is based on UML's **Activities** (p. 283). The mapping from structured text is quite straightforward here. Every SPO[PO] sentence maps into one **Action** (p. 280), while every conditional sentence maps into a **DecisionNode** (p. 319). All the scenarios in the text representation map to one or more **Activities** that contain **Actions** and **DecisionNodes** connected with **ActivityEdges** (p. 293). A single **Activity** (with an associated activity diagram) can therefore summarize a set of correlated scenarios contained in a use case.

Sequence graph representation is based on UML's **Interactions** (p. 419). This representation is distinct from the other notations in that it might reveal the internals of the developed system. For this reason this notation has to be used with care in order not to violate the basic characteristic of the use case being a description of behavior observable to the actor. The transformation from text representation is trickier here, as we need to map sentence parts into appropriate UML constructs which is not straightforward. Moreover, the mapping depends on the mapping of the domain vocabulary and the domain class model into the design class model. When constructing the sequence graph representation, every SVO[PO] sentence can be mapped into several **Messages** (p. 428). An object in the textual scenario sentence can be mapped into several design **Classes** (p. 86) from the design model (the ones that map from vocabulary entry associated with this object). A verb in the sentence maps into one or more operations of the above **Classes**. For the mapping to be coherent, appropriate **Lifelines** (p. 427) that receive **Messages** should be associated with relevant **Classes**. It can be noted that this approach is in close relation to the idea of distributing responsibilities between objects found e.g. in [1].

4 Conclusions and future work

The proposed set of related notations was used in several software development projects. In all the projects, the notation has shown to be good means of communication between various project roles. It seems to fulfill most of the requirements imposed by the roles as described in Section 2. The users have reported very good comprehension of scenarios written using the SPO[PO] notation. The associated mappings allowed for instant estimations of the influence that the changes in requirements imply on the user interface design or the system architecture. This last advantage of the presented trans-

formations was acknowledged to be very important for efficient management of projects in changing environments.

The definition of formal transformations between use case notations brings use cases closer to the current trend of model-driven software development [11]. Transformations defined on the level independent of computations (CIM), allow for pushing the control over the development efforts up to the users themselves. By using the proposed notations and mappings it would be possible to trace platform independent and platform specific constructs back to the system's functionality postulated by the users. An additional important ability of the proposed notation is the promotion of reuse as postulated in [13]. Requirements written in the proposed form can be easily divided into reusable chunks. These in turn indicate prospective software components to be reused in whole or in part. This indication is much simplified with appropriate mapping set between the requirement chunks and appropriate architectural elements.

Research associated with model or notation transformations is always influenced by the problems with their automation. It takes considerable effort to keep scenarios written in the four notations synchronized by hand. Future work is therefore associated with developing of an appropriate tool to support the proposed notation. The tool would allow for creating scenarios in the proposed structured text notation and enable transformations to other UML-based notations. The first version of the tool was already developed for a subset of the proposed notation and having limited transformation capabilities [7]. The succeeding versions shall be integrated with a commercially available UML-based CASE tool. This would allow for testing of effectiveness of the proposed transformation methods in real software development projects.

References

1. Biddle, R., Noble, J., Tempero, E.: From Essential Use Cases to Objects. In: Constantine L. (ed.): forUSE 2002 Proceedings, Ampersand Press (2002)
2. Cockburn, A.: Structuring Use Cases with Goals. Journal of Object-Oriented Programming, Vol. 5, no. 10 (1997) 56-62
3. Cockburn, A.: Writing Effective Use Cases. Addison-Wesley (2000)
4. Constantine, L.L, Lockwood, L.A.D.: Structure and Style in Use Cases for User Interface Design. In: van Harmelen, M. (ed.): Object-Modeling and User Interface Design, Addison-Wesley (2001)
5. Gelperin D.: Precise Use Cases, LiveSpecs Software (2004) http://www.livespecs.com/
6. Graham, I., Task Scripts, Use Cases and Scenarios in Object-Oriented Analysis. Object-Oriented Systems, Vol. 3, No. 3 (1996) 123-142
7. Gryczon, P., Stańczuk, P.: Obiektowy system konstrukcji scenariuszy przypadków użycia (Object-Oriented Use Case Scenario Construction System). MSc thesis, Warsaw University of Technology (2002)
8. Hurlbut, R.R.: Managing Domain Architecture Evolution Through Adaptive Use Case and Business Rule Models. PhD thesis, Illinois Institute of Technology, Chicago, (1998) http://www.iit.edu/~rhurlbut/hurl98.pdf
9. Jacobson, I., Christerson, M., Jonsson, P., Övergaard, G.: Object-Oriented Software Engineering: A Use Case Driven Approach. Addison-Wesley (1992)
10. Kruchten, P.: The Rational Unified Process: An Introduction. Addison-Wesley (1999)
11. MDA Guide Version 1.0.1. Object Management Group (2003)

12. Somé, S.S.: Beyond Scenarios: Generating State Models from Use Cases. In: Scenarios and state machines: models, algorithms and tools. ICSE 2002 Workshop, Orlando, Florida (2002)
13. Śmiałek, M.: Global Reuse Strategy Based on Use Cases. OOPSLA 2000 Companion, Conference on Object-Oriented Programming, Systems, Languages and Applications, Minneapolis (2000) 49-50
14. Unified Modeling Language: Superstructure, version 2.0, Final Adopted Specification, ptc/03-08-02. Object Management Group (2003)