Extending UML 2.0 Metamodel for Complementary Usages of the «extend» Relationship within Use Case Variability Specification

Alexandre Bragança[1] and Ricardo J. Machado[2]
[1] *Dep. Eng. Informática, ISEP, IPP, Porto, Portugal,*
alex@dei.isep.ipp.pt
[2] *Dep. Sistemas de Informação, Universidade do Minho, Guimarães, Portugal,*
rmac@dsi.uminho.pt

Abstract

Software product lines and related approaches, like software factories, are starting to capture the attention of the industry practitioners. Nevertheless, their adoption outside the research community and big companies is still very restricted. We believe that model-driven approaches, like OMG's MDA, with proper tool support, can bring the advantages of product lines to a broader audience. For this to become a reality, model-driven methods should integrate requirements models into the software development process. In this paper, we discuss the semantics of use case relationships and their formalization using activity diagrams to support variability specification. Particularly, we propose an extension to the «extend» relationship that supports the adoption of UML 2.0 use case diagrams into model-driven methods. Our proposal results from our work with 4SRS (4 Step Rule Set), a model-driven method in which use cases are the central model for requirements specification and model transformation.

1. Introduction

Software product lines enable high productivity levels in software development through proactive intra-organizational reuse. Nonetheless, such approaches imply relative demanding methods and, as such, are difficult to implement in small and medium sized companies. Model-driven approaches promise to facilitate the adoption of these demanding methods because they provide high levels of automation. One well known example of such fusion of approaches is the Microsoft software factories initiative [1].

Requirements and analysis are crucial activities of all software development processes. In the case of product lines, their importance is higher because they guide the design of the reference architecture of the product line and all the other common artifacts. As such, model-driven engineering approaches for product lines should integrate models of these phases.

To fully integrate requirements into model-driven approaches the requirement model has to be formalized. In the case of UML 2.0 this means the formalization of use cases. In product lines, a vital concern is the specification of variability. Figure 1 presents types of alternatives (variability in action flows) that are common in the textual description of use cases. The UML 2.0 use cases metamodel does not support all these types of alternatives. This paper addresses this limitation and proposes an extension to the UML 2.0 metamodel to support model-driven methods with such requirements for variability modeling. For the formalization of the behavior of use cases we propose the adoption of activity diagrams. We used the UML 2.0 specification as described in [2].

The findings presented in this paper result from our work on the integration of requirement models into a model-driven method called 4SRS (*4 Step Rule Set*)[3]. The initial goal of 4SRS was to provide a method to help analysts and designers develop large object-oriented systems through the use of models and rules for model transformation.

In previous works we have presented the first experimental results of adapting the method for explicitly handle variability [4]. In this paper we complement our previous work by addressing the formalization of UML 2.0 use cases by extending its metamodel. Our work on 4SRS also addresses the transformation of these new use case models into components and classes, i.e., moving from the problem domain to the solution domain. Nonetheless, this issue is out of the scope of this paper.

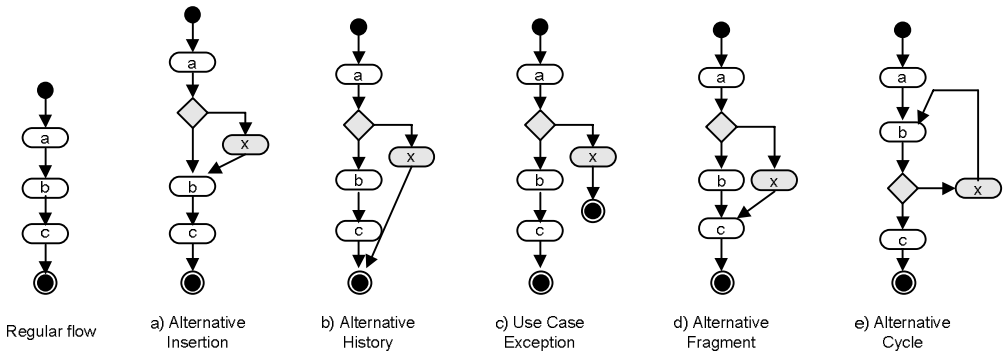

Figure 1. Types of alternative sequences of actions in use cases

The paper is structured as follows. The next section is dedicated to analyze the semantics of the major use case model elements regarding variability support and briefly describes our approach to specify behavior variability in use cases. In section 3 we present our proposal to the extension of the use case metamodel and how to formalize use case behavior through activity diagrams. In Section 4 we conclude.

2. Use Case Relationships

Regarding use cases, and according to the UML 2.0 metamodel, it is clear that in UML 2.0 only the *extend* relationship can be used to model variability. Nevertheless, other approaches have been proposed. For instance, Gomaa proposes that the *include* relationship can be used to model optional use cases in product lines [5]. In this paper we will only address variability in use cases through the *extend* relationship.

When developing software product lines, features and feature diagrams are also commonly used to model variability. Features represent user-visible aspects or characteristics of a domain [6]. When they represent functional characteristics of a product line they can be related to use cases. Usually a feature can be modeled by one or more use cases [5, 7]. Although the 4SRS method also integrates features diagrams, this paper does not address the relation between use cases and features.

Usually use cases are described in natural language. In fact, there is a pattern for the textual description of use cases that is generally accepted by practitioners [8]. In this pattern, use cases are composed by sequences of steps, or actions. There is usually one main sequence and many alternative sequences. There are five types of alternative sequences: conditional insertion; use case exception; alternative history, alternative part and alternative cycle [9]. Figure 1 presents a graphical representation of the possible sequence alternatives.

Use case **Renew Loan**:
- Main flow:
1. The Librarian enters the renew loan data (user ID and Item ID)
2. The system retrieves loan info
3. The loan info is displayed to the librarian
4. The system retrieves item info «extension point»
5. The system renews the loan «extension point»
Use case ends

- Alternative flows:
2a. Loan does no exist (after step 2)
 2a1. The system displays a message to librarian
 Use case ends
3a. A fine is due (after step 3)
 3a1. The librarian collects the fine «extension point»
 Use case rejoins (before step 4)
 Alternative flows:
  3a1a The fine is not totally paid (after step 3a1)
   3a1a1. The system displays a message to the librarian
   Use case ends
4a. The item is reserved (after step 4)
 4a1. The system displays a message to the librarian
 Use case ends

Figure 2. Excerpt of use case *Renew Loan*

An alternative insertion (Figure 1a) is used to represent conditional behavior that is inserted into a precise point (extension point) of a flow. In this case the insertion point is coincident with the rejoin point, i.e., at the end of the alternative behavior the flow rejoins the main flow at the initial extension point. This is very similar to an *include* relationship with a condition of insertion. Alternative insertions can be easily modeled by *extend* relationships because the extension point and the rejoin point are coincident. In contrast, the other types of alternatives (alternative history, use case exception, alternative fragment and alternative cycle) are not directly supported by the UML 2.0 use case metamodel (see Figure 1). This is an important limitation since in practice it is not so

unusual for extensions to have flows that are diverse from that of an alternative insertion.

Use case **Handle Gold Member**:
- Main flow: <empty>

- Alternative flows (Extension flows):
1. Handle Renew Loan
  Condition: MemberType=GoldMember
  1a. Handle Collect Fine
    (before *Librarian collects the fine*):
    1a1. If fine<member fee Rejoin base use case
      (before *Retrieve item info*).
    Rejoin base use case (before *Librarian collects the fine*).
  1b. Handle Borrow Rule
    (after *Retrieve item info*):
    1b1. If Item Reserved by non-gold member Rejoin
      base use case (before *Renew loan*)
    1b2. Display a message to the librarian
    Base use case ends
Referenced *Extension Points*:
-*Librarian collects the fine*:
  Renew Loan.The librarian collects the fine
-*Retrieve item info*:
  Renew Loan.The system retrives item info
-*Renew loan*: *Renew Loan.The system renews the loan*

Figure 3. Excerpt of use case *Handle Gold Member*

Lets consider the example of a library system and two use cases of that system, as presented in Figure 2 and Figure 3. In Figure 2 there are 2 types of alternatives: exceptions (2a, 4a and 3a1a) and alternative flow (3a). Figure 3 presents an excerpt of the description of the extending use case *Handle Gold Member*. This excerpt contains only the extending behavior that regards use case *Renew Loan*. *Handle Gold Member* extends *Renew Loan* and uses the three extension points defined in *Renew Loan*. Figure 3 presents common situations that reflect two types of alternatives that are not adequately handled by the *extend* relationship of UML 2.0:

- The extending use case adds conditional behavior that can result in an alternative flow (*1a. Handle Collect Fine* and *1b. Handle Borrow Rule*), i.e., there are rejoin points that do not match the original extension point;

- The extending use case adds conditional behavior that can result in an alternative history (*1b. Handle Borrow Rule*), i.e., the new behavior can lead to an alternative ending in the base use case.

As presented, the UML 2.0 metamodel only supports alternative insertion extensions (Figure 1a). This represents a major limitation to the modeling of the diverse variability types that are commonly specified by textual use cases. In the next section we present and discuss a proposal of an extension to the use case metamodel that addresses the modeling requirements identified in this section.

3. Extending the UML 2.0 Metamodel

In the past, several proposals have been made to formalize use cases [10-13]. Some recent works also proposed approaches to manage variability in use cases in the context of product lines [14, 15]. The main concern of their authors has been the lack of formalism of the usual use case text descriptions. Most well known proposals also regard non-visual languages. In our specific case we aim at integrating requirements into a model-driven method. In the context of UML 2.0, the modeling of behavior can be addressed by activity diagrams, so we have adopted activity diagrams for modeling use case behavior. Figure 4 presents an excerpt of the UML 2.0 metamodel adapted (extended) to support our proposal for formalization of use cases.

Figure 4 presents in gray metamodel elements that are extensions to UML 2.0. Since, according to UML 2.0 specification, a use case is a specialization of a *BehavioredClassifier*, we use the *classifierBehavior* and *ownedBehavior* associations to model, respectively, the use case main flow and the alternative flows. We propose a new *ExtensionFragment* metaclass to support the issues identified in the previous section. In our proposed metamodel an *extend* relationship can have a condition and make several extensions to a base use case. Each extension has one extension location but can have several rejoin locations. An extension also specifies which behavior of the extending use case will extend the base use case in the extension location. Since use case behaviors are formalized through activity diagrams, extension locations and rejoin locations refer to elements of type *Action* of the corresponding behavior. To clarify if the extension or the rejoin points are made before or after the corresponding *Action*, we propose the attribute *moment* in the *Location* metaclass. We also propose the new *InclusionPoint* element only to have a similar approach in *extend* and *include* relationships.

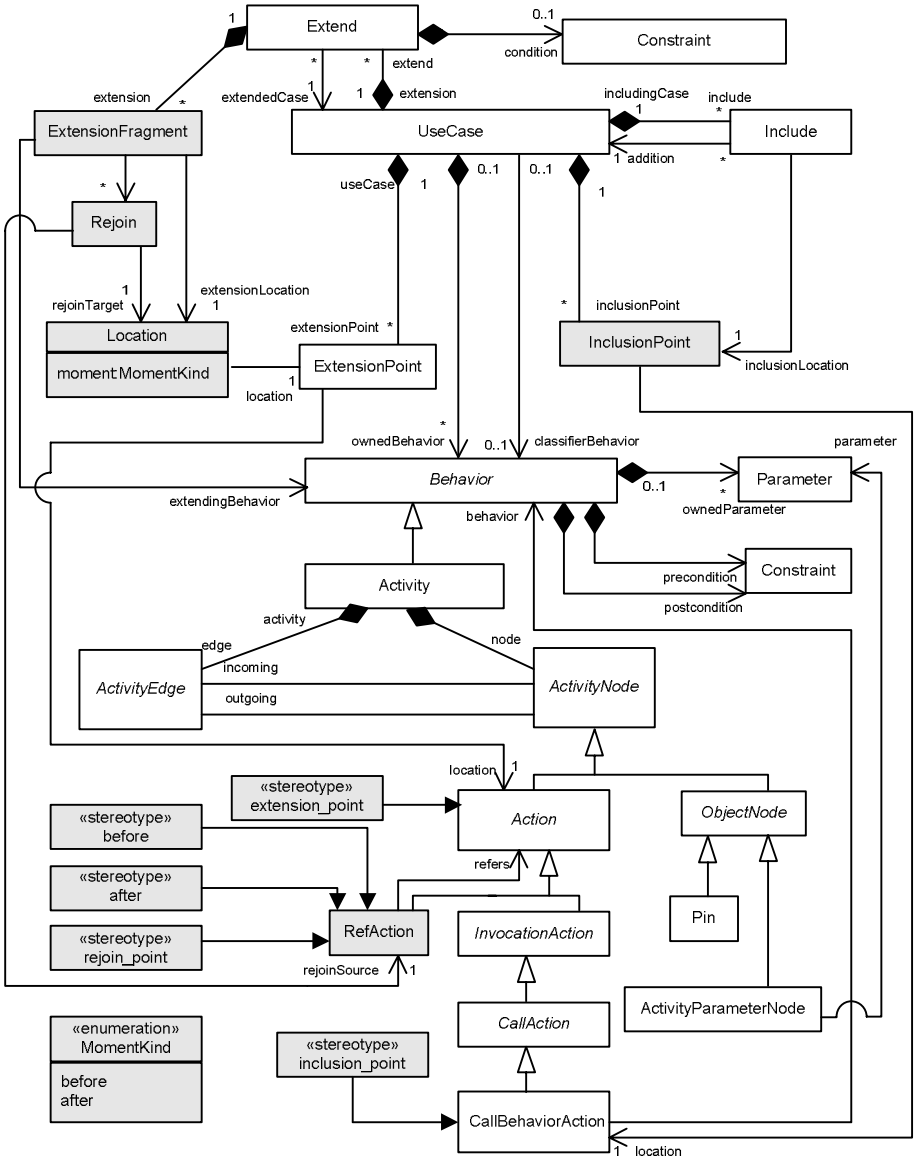

Figure 4. Excerpt of proposed metamodel

An *InclusionPoint* refers to the location where the behavior is to be included. This location has to refer to an element of type *CallBehaviorAction* of the same use case as the *include* relationship. It is not necessary to specify what behavior is to be included because the semantic of the *include* is to include the main behavior (*classifierBehavior*) of the included use case.

The stereotypes *extension_point*, *inclusion_point*, *rejoin_point*, *before* and *after* are used as a visual aid to identify more easily the semantics of the actions nodes of the activity diagrams.

Figure 5 presents the *extend* relationship between *Renew Loan* and *Handle Gold Member* use cases and follows our proposal for a default and simplified visual representation of the *extend* relationship.

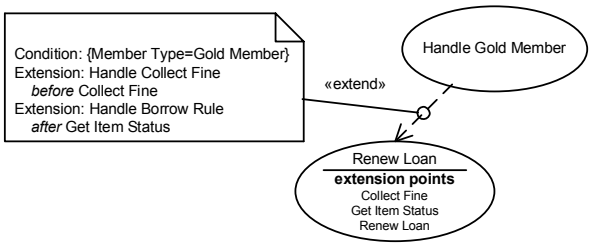

Figure 5. Proposed notation for the *extend* relationship

This visual representation only differs from the actual notation of UML 2.0 in the contents of the note attached to the *extend* relationship, since it reflects the new *ExtensionFragment* element of the metamodel (see Figure 4).

In this section, we have briefly described the major characteristics of a proposed metamodel to support the formalization of use case models with the aim of supporting their integration into model-driven methods. A complete discussion of the metamodel and related specifications, such as constraints, is out of the scope of this paper. These topics will be addressed in a future publication.

4. Conclusions

A generalized adoption of product line approaches can only become a reality if supported by model-driven methods. In order to accomplish this goal, model-driven methods should incorporate support for all phases of the software development process, including requirements. In this paper we have proposed an extension to the UML 2.0 metamodel in order to support this goal. Our proposal is based on UML 2.0 use case models and on the previous work of the authors on a model-driven method called 4SRS. The actual UML 2.0 metamodel has restrictions regarding the *extend* relationship: it only supports *alternative insertions*. We have proposed a complementary extension to the UML2 metamodel since it adds support for new types of alternative flows.

The formal specification of use cases provides an effective way to maintain the trace to requirements and enables model-driven development methods in which requirements models are first-class citizens. One could argue that a similar result could be achieved only through the UML profile extension mechanism. Even if this could be true, in our opinion, such approach would not promote the full inclusion of use case models into model-driven methods. To achieve this goal we advocate the adoption of metamodeling tools.

As described in the paper, the 4SRS method has attained a significant maturity level. Nevertheless, the management of variability truly raises the level of complexity of model-driven methods. For these methods to be adopted they must be supported by public available tools. Our ongoing work is to provide such support with the GME metamodeling toolkit [16]. We plan on presenting the results of our ongoing work in the near future.

5. References

[1] Greenfield, J., K. Short, S. Cook, and S. Kent, *Software Factories: Assembling Applications with Patterns, Models, Frameworks, and Tools*, Wiley, 2004.

[2] OMG/UML, *Unified Modeling Language: Superstructure*, OMG, Version 2.0, formal/05-07-04, 2006.

[3] Machado, R.J., J.M. Fernandes, P. Monteiro, and H. Rodrigues, "On the Transformation of UML Models for Service-Oriented Software", In *ECBS International Conference and Workshop on the Engineering of Computer Based Systems*, Greenbelt, Maryland, 2005.

[4] Bragança, A. and R.J. Machado, "Deriving Software Product Line's Architectural Requirements from Use Cases: an Experimental Approach", In *2nd International Workshop on Model-Based Methodologies for Pervasive and Embedded Software*, Rennes, France, 2005.

[5] Gomaa, H., *Designing Software Product Lines with UML*, Addison Wesley, 2005.

[6] Kang, K.C., S.G. Cohen, J.A. Hess, W.E. Novak, and A.S. Peterson, *Feature-Oriented Domain Analysis (FODA) Feasibility Study Technical Report*, Software Engineering Institute, Carnegie Mellon University, CMU/SEI-90-TR-21, 1990.

[7] Griss, M.L., J. Favaro, and M. d'Alessandro, "Integrating Feature Modeling with the RSEB", In *Fifth International Conference on Software Reuse*, Victoria, Canada, IEEE Computer Society Press, 1998.

[8] Cockburn, A., *Writing Effective Use Cases*, Addison-Wesley, 2001.

[9] Metz, P., J. O'Brian, and W. Weber, *Specifying Use Case Interaction: Clarifying Extension Points and Points of Rejoin.* Journal of Object Technology, (March/April 2004), 2004.

[10] Overgaard, G. and K. Palmkvist, "A Formal Approach to Use Cases and Their Relationships", In *«UML»98: Beyond the Notation*, Ecole Superieure des Sciences Appliques pour l'Ingenieur - Mulhouse, Universite de Haut-Alsace, France, 1998.

[11] Stevens, P., "On Use Cases and Their Relationships in the Unified Modelling Language", In *FASE'01*, Springer, 2001.

[12] Porres, I., *Modeling and Analysing Software Behavior in UML,* PhD, Abo Akademi University, 2001.

[13] Hurlbut, R., *Managing Domain Architecture Evolution Through Adaptive Use Case and Business Rule Models,* PhD, Illinois Institute of Technology, 1998.

[14] Fantechi, A., S. Gnesi, G. Lami, and E. Nesti, "A Methodology for the Derivation and Verification of Use Cases for Product Lines", In *SPLC 2004*, Springer, 2004.

[15] Eriksson, M., J. Borstler, and K. Borg, "The PLUSS Approach - Domain Modeling with Features, Use Cases and Use Case Realizations", In *SPLC 2005*, Springer-Verlag, 2005.

[16] Ledeczi, A., M. Maroti, A. Bakay, G. Karsai, J. Garrett, C. Thomason, G. Nordstrom, J. Sprinkle, and P. Volgyesi, "The Generic Modeling Environment", In *WISP'2001*, Budapest, Hungary, IEEE, 2001.