Modeling Variability by UML Use Case Diagrams

Thomas von der Maßen, Horst Lichter
{vdmass, lichter}@cs.rwth-aachen.de
Research Group Software Construction
RWTH Aachen

Abstract

Software Product Lines are characterized through common and variable parts. Modeling variability is one of the most important tasks during the analysis phase. In order to guarantee that domain experts and developers understand each other variability has to be modeled explicitly. In this paper we present and examine the different types of variability. We analyze and assess UML Use Case diagrams in detail with respect to their capability to express variability, with the result, that they only provide poor assistance. Consequently we introduce an enhancement of the Use Case meta-model to overcome this deficiency by integrating new modeling elements that allow to express the identified types of variability.

1 Introduction

Modeling variability is an essential task in developing Software Product Lines [PLP01]. Variability means that beside the common parts that are shared by all members of a Software Product Line, each product has its own specific parts. Defining the commonality and the variable parts of a Software Product Line is mainly done during product line scoping [Sch00]. But, variability exists and must be modeled in every phase of the product line development process which means in the requirements analyzing and in the design phase. Variability is finally realized in the implementation phase.

Besides considering variability in different phases it is very important to take into account that different types of variability exist. Halmans [Hal02] differentiates between *technical variability*, comprising all kinds of variability that exist in the system infrastructure, concretion and realization of the product line and *functional variability*, defining functional and quality characteristics of the system. Technical variability is defined in terms of “how” a product line can be implemented; functional variability is defined in terms of “what” the product line should be capable of.

The analysis of functional variability is necessary to identify characteristics that are mandatory, that means that they are common, and characteristics that are variable, that means, they are not mandatory but can be considered in a specific context or not. To communicate and negotiate common and variable characteristics with the stakeholders an appropriate notation must be chosen to guarantee that domain experts and developers understand each other.

In the next section we describe which types of variability exist and list requirements on a notation for modeling variability. Section 3 gives a brief overview on the feature based modeling approach and considers UML Use Cases in the context of modeling functional variability. In section 4 we present an extension of the UML Use Case meta-model, integrating variability-modeling elements. Finally, we draw some conclusions and give a brief outlook on our future research.

2 Modeling Functional Variability

2.1 Types of variability

Variable system characteristics are defined by means of so called *variation points* (see [Jac97]). At a variation point different specific variants can be chosen for each family member to resolve this variation point. The following types of functional variability must be considered:

- **Options**
  Optional characteristics of a system can be integrated or not. That means from a set of optional characteristics, any quantity of these characteristics can be chosen, including none or all. We distinguish between options that can only be chosen if a specific condition holds and options where one has a free choice to integrate them or not. Hence, optional aspects can be modeled by means of an or-relationship.
- **Alternatives**
  From a set of alternative characteristics, only one characteristic can be chosen - defining an exclusive-

or/xor-relationship, which means a “1 from n choice”. Again, we distinguish between alternatives that are linked to a specific condition, or not.

- **Optional alternatives**
  At last a combination of optional and alternative characteristics must be considered. This is the case, if at a variation point alternatives are available, but it can be chosen if these alternatives are relevant at all – that means a “0 or 1 from n choice”.

2.2 Levels of variability

Modeling variability can be done in different views and on different levels of abstraction. Whereas the level of abstraction determines the granularity of descriptions of characteristics, different views reveal information about perspectives on a system. In addition, variability can be examined in different scopes.

Figure 1 shows that variability exists among the different members of a product line designed for a specific domain. That means, the various members (products) provide different functionality. This variability is modeled at the domain level and is resolved when a concrete product is instantiated. Though the variability between the product line members is resolved, variability will occur for each product at runtime. Runtime variability should be modeled at the product level. It occurs if different application flows exists. The application flow is usually driven by the user or external systems.

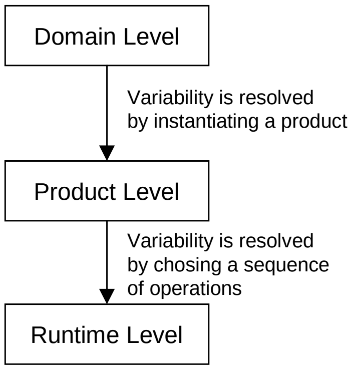
Domain Level
Variability is resolved by instantiating a product
Product Level
Variability is resolved by chosing a sequence of operations
Runtime Level

Figure 1: Resolving variability

2.3 Requirements on a modeling notation

To express variability it is necessary to model *variation points* and its variable parts [Jac97]. Modeling notations can be graphical, textual or a mixture of both. The advantage of modeling variability in a graphical notation is, that variation points are recognized much easier than in a pure textual description. In this section we present important requirements that have to be put on a notation for modeling variability. From the modeling point of view, the following requirements have to be considered:

- **Representation of common and variable parts**. It must be possible to express common and variable parts of a Software Product Line. Hence, distinct modeling elements are needed to explicitly express common characteristics belonging to the platform and variable characteristics. Variable characteristics should always be modeled in the context of a variation point.
- **Distinction between types of variability**. The notation must be able to explicitly express all different types of functional variability introduced in section 2.1.
- **Representation of dependencies between variable parts**. The description of dependencies is mandatory. Dependencies between variable parts are implied, equivalent and xor-relationships. An implied-relationship means, that if one characteristic is needed, than another characteristic must be taken into the system as well. An equivalent-relationship is an implied-relationship in both directions and a xor-relationship expresses that only one characteristic from a set of characteristics can be taken into the system.

Beside these modeling requirements the notation should support the following aspects:

- **Supporting model evolution**. Because models will evolve over time, new requirements and characteristics have to be integrated and the model must be changed easily.
- **Providing good tangibility and expressiveness**. To communicate the resulting models to the domain experts the models must be easy to read and to understand. In most cases domain experts are not able to understand formal textual notations. So a graphical notation might help to understand the relevant parts very easily. It is important to mention that the graphical notation should not replace natural language descriptions but to supplement them and to provide a different view on the same context.

3 Feature and Use Case Modeling

In this section two notations will be presented to model variability from different points of view: Feature graphs and UML Use Case diagrams. These two notations will be analyzed with respect to the requirements mentioned in section 2.3.

3.1 Feature graphs

The Feature Oriented Domain Analysis (FODA) is a common approach to model variability during domain analysis [Kan00]. In FODA, features are the central modeling element. A feature is defined as a "prominent or distinctive user-visible aspect, quality or characteristic of a software system or systems". The core of a feature model is a feature graph. A feature graph represents variability in a very compact and clear way, in that it presents the features in a tree of AND/OR nodes to identify the common and variable parts within the domain. Eisenecker describes the capability of feature graphs in detail [Eis00]. He points out, that a great advantage of feature graphs over UML class diagrams is that the representation is independent from implementation aspects and that features are easier building blocks than objects, because no mechanism like generalization, association or parameterization is used. The successful use of feature graphs is described in [Eis01] and [Phi00]. Figure 2 shows a small feature model in the domain of an automated teller machine, that provides several banking services and authorizes identification either through a chip card or fingerprint.

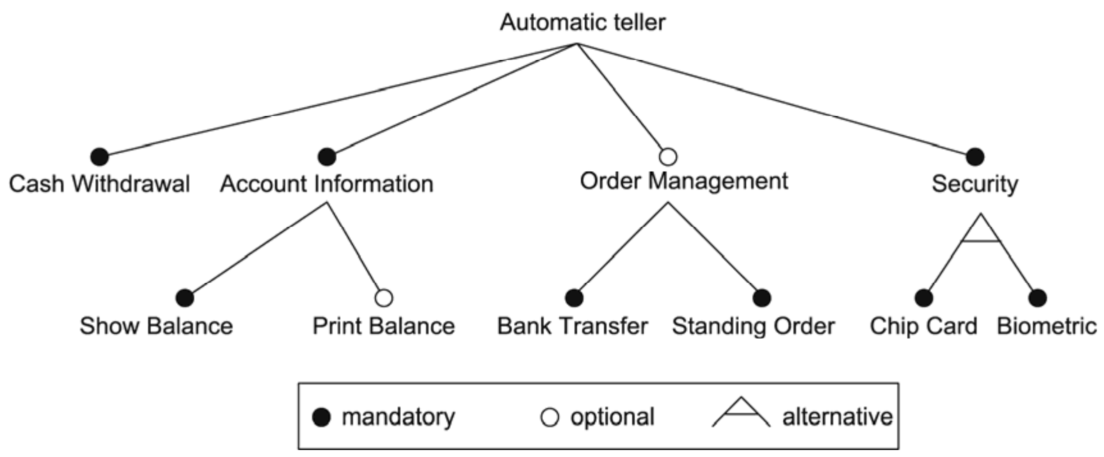

Figure 2: Example of a feature graph

A feature graph illustrates a structural view on a system. Characteristics of the system are structured hierarchically in a tree. A characteristic groups the functionality that constitutes this characteristic. Common characteristics can be modeled through mandatory features and variable parts can be modeled through optional or alternative features, which fulfill the requirements to distinguish between different types of variability. Therefore, feature graphs are able to express the identified variability, whereas conditions for optional or alternative types must be specified explicitly through a textual note. In addition, dependencies exist implicitly through the hierarchy of features, that means the existence of a child feature implies that the parent feature exists, as well. A xor-relationship exists between alternative features which have the same parent feature. Dependencies between features that don't have the same parent cannot be modeled directly in the feature graph, though it is possible to add composition rules that "supplement the feature model with mutual dependency and mutual exclusion relationships which are used to constrain the selection from optional or alternative features" [Kan02]. Unfortunately feature graphs and FODA are not widely known, accepted and used. Therefore, the practical use is constrained [Spe02].

In the next section we analyze UML use case diagrams with regard to their capability to model variability.

3.2 Use Case diagrams

During requirements analysis, Use Case diagrams help to identify the actors and to define by means of Use Cases the behavior of a system (see [Jac92]). Use Case modeling is accepted and widely used in industry. If Use Case diagrams support the modeling of functional variability, they can also be used to describe common and variable behavioral characteristics of a Software Product Line. Hence, we take a brief look on the UML Use Case modeling elements. Beside actors and Use Cases UML defines a small set of relationships to structure actors and Use Cases. Use Cases may be related to other Use Cases by the following relationships:

- **Extend**
  An extend relationship implies that a Use Case may extend the behavior described in another Use Case, ruled by a condition.
- **Include**
  An include relationship means that a Use Case includes the behavior described in another Use Case.
- **Generalization**
  Generalization between Use Cases means that the child is a more specific form of the parent Use Case. The child inherits all features and associations of the parent, and may add new features and associations [UML01].

It is obvious that it is possible to model options that are linked to a condition through the extend relationship. All other types of functional variability cannot be modeled directly with Use Case diagrams. To illustrate this we examine the following example in the context of the automatic teller machine, mentioned above. Using the automatic teller machine an authentication procedure is necessary to get access to the banking services. The authentication can be performed by inserting a chip card or by giving a fingerprint. The authentication procedure is predetermined by the teller machine, that means the teller machine provides only one authentication procedure. A Use Case diagram where extend-relationships are used to model options is depicted in figure 3. Unfortunately this diagram doesn't model the context correctly. It is not clear that exactly only one extension Use Case must be executed. The two extension Use Cases are mutually exclusive, so.

Though it is possible to introduce stereotypes to mark relationships between Use Cases, stereotypes typically lack of a defined semantic and may have different meanings in different contexts. Furthermore it is possible to define constraints between Use Cases, but their semantics may vary in different contexts, too.
Therefore new Use Case modeling elements are needed to explicitly express all types of variability described above. It should be mentioned, that feature graphs, which represent characteristics of the system, can be used to complement Use Case modeling and to organize the results of the commonality and variability analysis in preparation for reuse [PLP01].

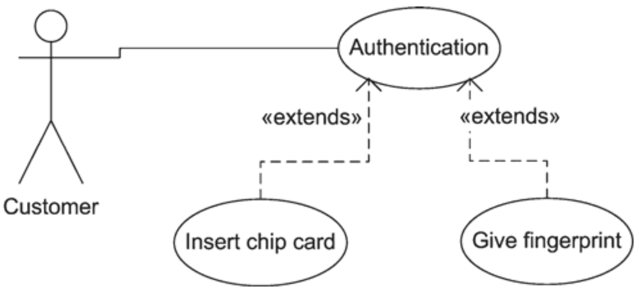

Figure 3: Modeling options by means of extend relationships

3.3 Comparison

These two notations allow to model variability in a domain from two different perspectives. Variability exists in features and in sequences of activities that means between product line members and at runtime of a specific product. Feature graphs are structure oriented because they describe the characteristics of a domain and the relationships between them. Furthermore they are intended to show hierarchies in structures and visualizes dependencies between characteristics, though in a restricted way because only dependencies between parent and child nodes can be expressed directly. Use Cases instead are suitable to model sequences of activities and they model therefore dynamic characteristics and dependencies between these activities.

A suitable process in capturing variability in an unknown domain is to start with capturing Use Cases to find essential activities that are executed in that domain. These essential activities can be structured further and can be extended on a more fine granular level. In a second step the variability in Use Cases can be summarized in features that can be modeled through feature graphs to identify common and variable characteristics. Thus feature graphs form a consolidation of variable sequences and performed activities.

To model variability in Use Cases the meta-model must be extended. This extension is introduced in the next chapter.

4 Extending the UML Use Case Meta-Model

4.1 Model extension

To explicitly model all types of variability described in section 2.1 the UML Use Case meta-model [UML01] has to be extended by two new relationships: *Option* and *Alternative*.

An option relationship between two Use Cases means that the behavior defined in the extending Use Case may be executed in the extension Use Case. This depends firstly on whether the functionality is provided by a potential product and not on a condition determined by the system or on earlier activities carried out by the actor. An optional behavior that depends on a condition can be modeled through the existing extend relationship.

An alternative relationship means that in the base Use Case exactly one alternative Use Case must be executed. Alternatives own an attribute *choice*, that consists of pairs of conditions and linked Use Cases. The conditions define under which circumstances an alternative Use Case is executed. The choice, which alternative to use may depend on a condition or not. The condition is formulated in a Boolean expression. This relationship exists always between one base Use Case and at least two alternative Use Cases. These new relationships can be integrated easily in the UML meta-model. Figure 4 shows the resulting meta-model.

To model variation points, an additional new model element *VariationPoint* has been included as a specialization of *ExtensionPoint*. A VariationPoint additionally has an enumeration of the alternative Use Cases. Alternatives and options are new relationships to express explicitly these types of variability.

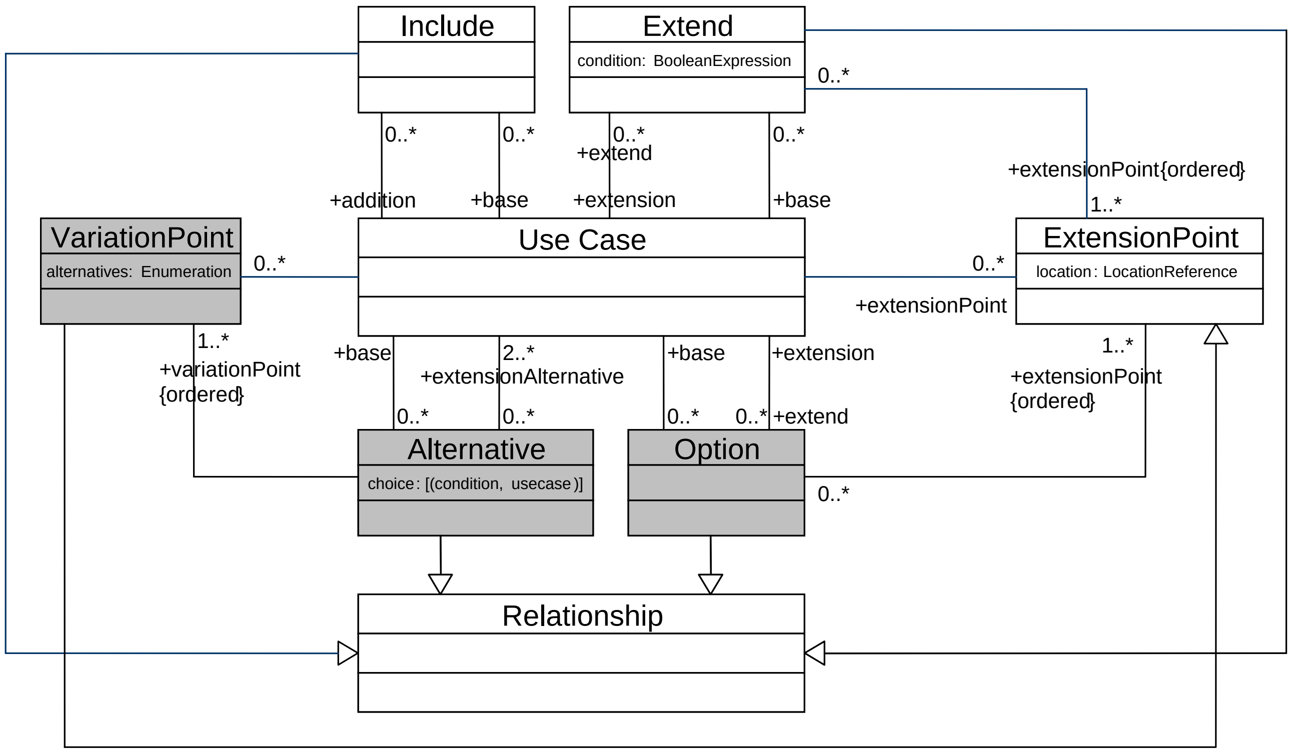

Figure 4: Extended Use Case meta-model

Table 1 shows the types of variability and the corresponding use case modeling element.

Type of variability	**Modeling element**
Options, linked to a condition	Extend relationship
Options, free choice	Option relationship
Alternatives, linked to a condition	Alternative relationship
Alternatives, free choice	Alternative relationship
Optional alternatives	Combination of optional and alternative relationships

Table 1: Modeling variability

Like in feature graphs, dependencies between variable and common parts are modeled implicitly. An including Use Case implies that the included Use Case is executed, too. A xor-relationship exists between alternative Use Cases that are part of the same variation point. In figure 5 an updated Use Case diagram for providing different forms of authentication is illustrated that uses the new alternative relationship to model the context correctly. Figure 6 shows the optional possibility to print the account balance.

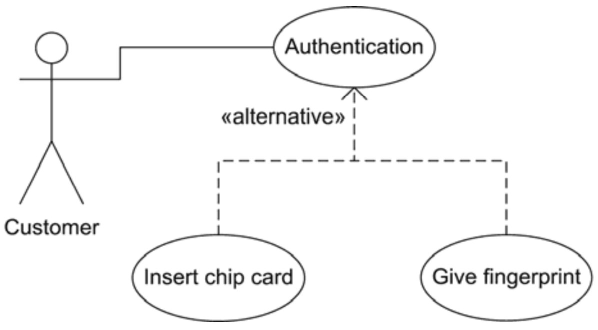

Figure 5: Example of an alternative relationship

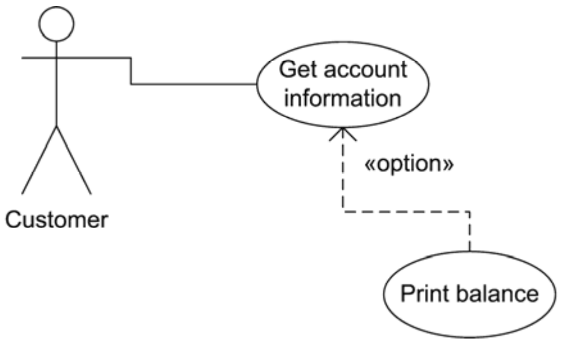

Figure 6: Example of an optional relationship

4.2 Domain variability vs. runtime variability

Considering the new variability modeling elements in the extended Use Case meta-model, the modeler is assigned to use them appropriately. That means the new modeling elements can be used on the one hand to express variability in the domain and therefore between members of the product line and on the other hand to express runtime variability (see section 3). A typical example of modeling runtime variability is illustrated in figure 7.

In this example the *Select order* procedure can be accomplished by to alternatives: either, the customer can choose to execute a bank transfer or a standing order – the customer is free of choice but can only perform one procedure at a time. Since both alternatives should be available in all applications in the domain of an automatic teller machine, this alternative relationship does not model variability between products. Though the new modeling elements give the modeler the choice to model runtime variability or variability between members of a product line, whereas a mixture of both should be avoided.

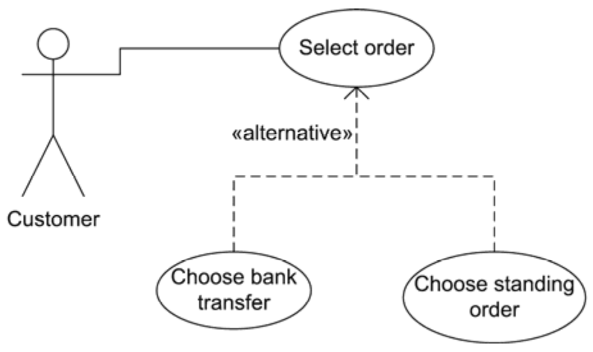

Figure 7: Modeling runtime variability with Use Case diagrams

5 Conclusion and future work

This study points out that one of the main tasks in requirements engineering for Software Product Lines is to identify and to express functional variability. This paper shows that several types of variability exist and have to be modeled. Therefore we examined briefly the feature graphs from FODA and in detail the UML Use Case diagrams with regard to their capability to express these types of variability.

Because UML Use Case diagrams are not able to model all kinds of variability, the meta-model had to be extended to fulfill the requirements that have been elicited. With the additional modeling elements the Use Case diagrams were able to express all identified types of variability with a defined semantic and in a readable and understandable way. Dependencies between Use Cases are modeled implicitly through include and alternative relationships.

It should be mentioned that Use Cases are only capable of modeling system behavior. They are not able to express static structures and characteristics of systems. Therefore a combination of feature graphs and extended Use Case diagrams can help to understand the complexity of extensive domains, processes and systems.

Our future research will be placed in validation and gaining experiences in praxis using the new modeling elements. Furthermore we are going to analyze dependencies between Use Cases that don't have direct relationships. The studies will show, whether it is necessary to consider these dependencies at all and if so, how they can be modeled in the Use Case diagrams. As a second step, we want to analyze in which way feature graph and Use Case modeling cohere and can be combined to help understanding complex processes.

References

[Eis00] Ulrich Eisenecker, Krzysztof Czarnecki. *Generative Programming – Methods, Tools, and Applications*. Addison-Wesley 2000.

[Eis01] Ulrich Eisenecker et. al. *Merkmalmodellierung für Softwaresystemfamilien*, in: OBJEKTspectrum 5/01, p. 23-30.

[Hal02] Günter Halmans, Klaus Pohl. *Modellierung der Variabilität einer Software-Produktfamilie*, Proceedings Modellierung 2002, p. 63-74. Lecture Notes in Informatics, 2002.

[Jac92] Ivar Jacobson. *Object-Oriented Software Engineering - A Use Case Driven Approach*. Addison-Wesley, 1992.

[Jac97] Ivar Jacobson, Martin Griss, Patrik Jonsson. *Software Reuse – Architecture, Process and Organization for Business success*. Addison-Wesley, 1997

[Kan00] Kyo Kang et al. *Feature-Oriented Domain Analysis (FODA) Feasibility Study*. Technical report CMU/SEI-90-TR-021, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, 2000.

[Kan02] Kyo Kang et al. *Concepts and Guidelines of Feature Modeling for Product Line Software Engineering*. 7th International Conference on Software Reuse (ICSR), Austin, Texas, USA, pp. 62-77, April 15-19, 2002.

[Phi00] Ilka Philippow, Kai Böllert. *Erfahrungen bei der objektorientierten Modellierung von Produktlinien mit FeatuRSEB*. Technische Universität Ilmenau, 2000. Available at: http://www.theoinf.tu-ilmenau.de/ pld/pub/index.html.

[PLP01] Paul Clements, Linda M. Northrop. *A Framework for Product Line Practice - Version 3.0*. Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2001. Available at: www.sei.cmu.edu/plp/framework.html.

[Sch00] Jean-Marc DeBaud, Klaus Schmidt. *A Systematic Approach to Derive the Scope of Software Product Lines*. Fraunhofer Institute for Experimental Software Engineering (IESE), Kaiserslautern 2000.

[Spe02] Andreas Speck, Matthias Clauss, Bogdan Franczyk. *Concerns of Variability in „bottom-up" Product-Lines*. Proceedings of the Workshop on Aspect-Oriented Software Development 2002.

[UML01] *OMG Unified Modeling Language*, Version 1.4, September 2001. Available at: http://www.uml.org.