

Modeling variability with UML

Matthias Clauß

Intershop Research Software Engineering Group
Intershop, Jena Dresden University of Technology
Matthias.Clauss@gmx.de

Keywords: product families, domain modeling, feature modeling, variation points, UML, UML profile

Classification: (parts of) diploma thesis

1 Introduction

Modeling variabilities and commonalities is a key element in developing product families and product lines. The objective of the analysis of commonalities and variabilities is strategic reuse – to identify common parts and develop them for reusability. The variability in such a domain (or several domains) distinguishes the members of a family from each other and needs to be explicitly modeled and separated from the common parts. But up until now, there is no uniform and consistent notation for modeling variabilities and commonalities.

On the other hand, the UML is the most well-known and accepted notation for modeling a software system. It describes a notation, specified by a metamodel, semantics and rules. Presently it is able to model a wide range of software systems, but it is inappropriate for modeling groups of related systems necessary for product families.

This paper introduces a UML extension to support feature diagrams and adds elements describing variability in the standard kinds of UML diagrams. The extensions use the standardized extension mechanisms and are therefore fully compatible with the standard and enable a broader usability.

Due to space limitations some interesting parts has to be shortened or omitted here — these can be found at [11].

2 Variability modeling in UML

This paper sketches a proposal for modeling variabilities in software families with UML. It is realized using the standardized extension-mechanisms of UML and can become part of UML in form of a UML profile.

It introduces three extensions: one for feature models and two smaller extensions for modeling selection-specific variability in software.

The feature modeling extension is strictly speaking a workaround to introduce feature diagrams (as proposed in [2, 6, 8]) into UML. It is needed to provide a way to systematically organize the variabilities of a group of software systems.

Secondly, an extension for the explicit representation of variation points (used in [5]) is specified. This shows variability - especially alternatives - in the analysis and design of software.

The third extension comprises optional elements provided for situations where variation points can not be used or are not applicable, e.g. in operations, attributes, associations or even behavioral diagrams. Variation points and optional elements can be used to design the variability expressed in the feature model.

A more detailed discussion of modeling feature interactions with this extensions took place in [10].

2.1 An extension for feature modeling

The most general definition of a feature is: a feature is a recognizable characteristic of a system relevant to any stakeholder [4].

Figure 1 shows an example from the eCommerce domain of Intershop that should demonstrate the usage of all primary modeling elements.

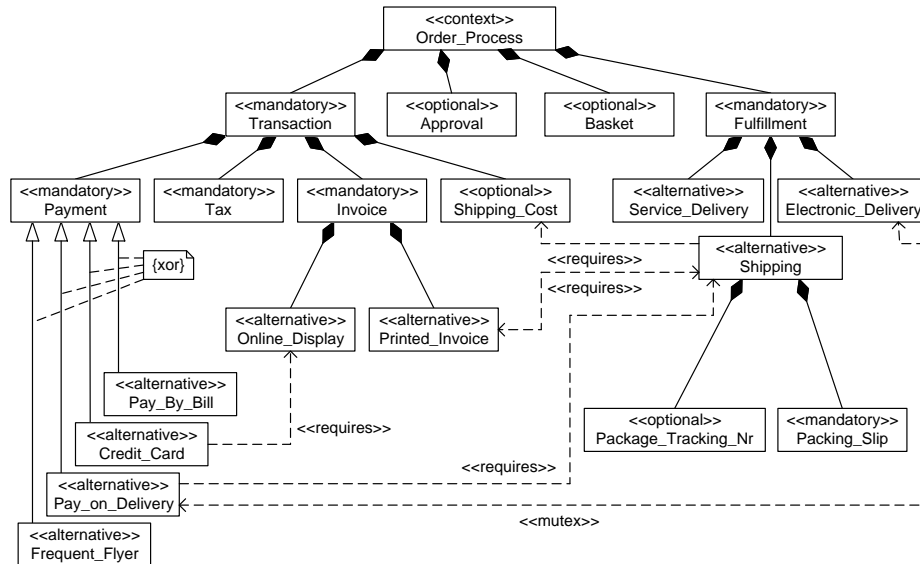


Fig 1. Sample feature model using UML

Four types of features are distinguished: mandatory, optional, alternative and external (a feature realized by the underlying platform, not by the system itself). The former three types are defined in FORM [6] respective FODA, external features are used in [8] and seem to be useful for describing relations to external (e.g. platform) requirements.

The features are organized in a tree with the modeled concept as the root constructed from composition or generalization relationships. The type of relationship to use depends on the identified kind of the relation between features. These relationships are modeled with their UML notation – filled diamond and generalization-arrow. Exclusive

and non-exclusive alternatives (or-features, [4]) can be distinguished by annotating the former with a {xor}-constraint (like that for associations in UML).

Composition rules are modeled in form of cross-tree constraints. Currently we use mutual exclusion between features ('mutex') and 'requires'-dependencies as hard constraints and think about the necessity of weak constraints to explicitly indicate possible dependencies that do not appear in all contexts (found as conflicts in [7]). According to the semantics of dependencies in UML constraints can be distinguished between uni- or bidirectional.

In addition, every feature node can be described with additional attributes, e.g. reasons for its selection (rationale). If desired, each node can be completed with the specification of a binding time, chosen from {development, installation, runtime}. Such additional properties are notated as tagged values.

More complex feature models as used in [9] are yet not supported. If required all ways are open to support such special requirements and to adapt this extension of UML.

2.2 Variation points

The notion of variation points can be applied to all classifiers and generalizable elements in UML. This includes use cases, packages, components, classes and collaborations.

The extension for variation points consists of two parts: an explicit marking of the location of a variation and the distinction of different ways to bind this variation (named variant's). The location is marked with the stereotype «variationPoint» and contains several attributes to describe the variation in detail. The mandatory connection from each variant to its variation point is modeled with the implementation specific technique, e.g. generalization, template binding or if not yet specified, with dependencies.

The selection of a variant can be determined using a condition that evaluates a boolean value, the variation point determines when (binding time) and how many variants (multiplicity) can be bound (i.e. selected for the desired product).

In addition, constraints for the variants can be modeled. At the moment a distinction into 'mutex', 'requires' and 'evolution' is used. The hard constraints ('requires' and 'mutex') are the same as those used for feature modeling. The last one is intended to support the evolution of the modeled systems and can be further separated; for example, into 'replaces', 'decomposites' or 'extends'. These constraints can be applied between variants and other modeling elements (including other variants), but not between a variant and its variation point. More general, these constraints can also be used for optional elements.

In opposite to [5], a variation point is modeled in the modeling element where the variation occurs because it is not an extra entity (in contrast to a class or component). An impact of this is the need for more than one variation point in a single model element. For example, a sorting component can be varied in the algorithm and the data types – these are two independent variations and should be explicitly distinguished in the model. For these purposes more than one variation point can be specified per modeling element. Such multiple variation points are included but not further discussed here due to space reasons.

Figure 2 shows the sample use of variation points and optional elements (section 2.3) in UML. The notation can also contain additional tagged values for special purposes, e.g. properties for binding mode, selection state, and others.

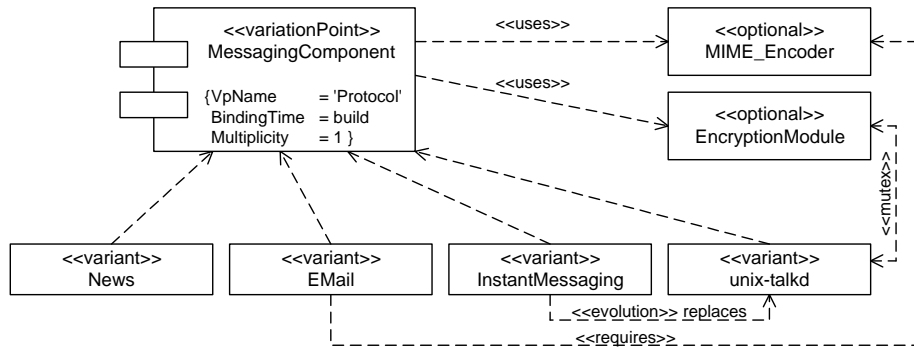


Fig 2. Sample usage of variation point and optional elements

2.3 Optional elements

Sometimes the notion of variation points is not applicable, e.g. on operations or attributes. For these cases optional elements can be used. An «optional»-Stereotype simply describes a model element that exists under some circumstances and in others not (related to a product family). This extension is simple, and can be applied on all UML model elements - even in behavioral diagrams if desired.

Every optional element has a binding time and a condition that determines its existence. In comparison to feature models, the binding time is a bit more extended, the proposed values are {development, build, installation, start-up, runtime} — the same as for variation points. The condition for optional elements and variants is omitted in the example.

2.4 Integration of the extensions

The usage of these extensions of course depends on the development process. As UML is used feature modeling can be seamlessly integrated with other models, e.g. use case models. To establish links between models several ways can be used: constraints between variants can be applied between variants/optional elements and other model elements to indicate interdependencies with other system parts. Secondly formal expressions in variants and optional elements describe decision rules for their selection (tagged value Condition). These shall also be used in conjunction with the feature model – to form expressions with the existence of features as an argument. As third «trace»-dependencies already exist in UML and provide forward- and backward-traceability.

3 Summary

The extensions for modeling variabilities in UML pick up two important concepts from product family engineering: feature modeling and the notion of variation points. The latter is complemented with optional elements possessing a broader applicability. For forthcoming requirements or special purposes, the notation can be adapted very easily using the standardized extension mechanisms.

It enables the modeling of variabilities in the UML — as needed for domain engineering. For these purposes it extends the broadly usable modeling notations specified in UML and adds feature modeling capabilities for the systematic management of software families.

This approach allows to extend feature modeling by UML typical elements, e.g. the annotation of (OCL-)constraints to formally specify composition rules or the easy use of «trace»-abstractions to other models. It enables the standard-conform use of UML for domain engineering and permits the seamless integration of different models.

The proposed extensions are specified to resemble a UML profile, primarily based on version 1.4, but backwards-compatible to version 1.3. The detailed discussion, especially finding elements with matching semantics, is part of the underpinning diploma thesis ‘Modeling variability with UML’, soon available in the German language [11].

Many details are subject to discussion – from a UML view and from the viewpoint of the product line community. Nevertheless, the objective is to specify a general and standardized extension of UML that is usable in mostly all existing product family/product line-approaches. This should improve tool support and enable an easier usability of the concepts that would enable systematic reuse thereby improving the development of software.

References

- [1] Object Management Group (OMG), *OMG Unified Modelling Language Specification*, versions 1.3 and 1.4 Draft, March 2000 / February 2001
- [2] K. Kang et al, *Feature-Oriented Domain Analysis (FODA) Feasibility Study*, Technical Report No. CMU/SEI-90-TR-2, November 1990
- [3] M. Griss, J. Favaro, M. d’Alessandro, *Integrating Feature Modelling with the RSEB*, International Conference on Software Reuse, June 1998
- [4] K. Czarnecki, U. Eisenecker, *Generative Programming – Methods, Tools and Applications*, Addison-Wesley, 2000
- [5] M. Coriat, J. Jourdan, F. Boissourdin, *The SPLIT method*, In Proceedings of the First International Software Product-Line Conference (SPLC-1), August 2000
- [6] K. Kang, et. al., *FORM: A Feature-Oriented Reuse Method with Domain-Specific Reference Architectures*, In Annals of Software Engineering 5, 1998
- [7] J. Bosch, *Evolution and Composition of Reusable Assets in Product-Line Architectures: A Case Study*, In Proceedings of the 1st Working IFIP Conference on Software Architecture, February 1999
- [8] M. Svahnberg, J. van Gorp, J. Bosch, *On the notion of Variability in Software Product Lines*
- [9] A. Hein et al, *Applying Feature Models in Industrial Settings*, Proceedings of the First International Software Product-Line Conference (SPLC-1), August 2000
- [10] M. Clauß, *A proposal for uniform abstract modeling of feature interactions in UML*, FICS-workshop, 15th European Conference on Object-Oriented Programming (ECOOP’01), April 2001
- [11] M. Clauß, *varUML sources: Full list of references for diploma thesis*, <http://www-st.inf.tu-dresden.de/~mc3/varUML>