Generic Modeling using UML extensions for variability

Matthias Clauß

Intershop Research Software Engineering Group
Intershop, Jena Dresden University of Technology
M.Clauss@intershop.com

September 14, 2001

Abstract

The abstract modeling of variability between members of a product family is necessary for development with reuse and realized using variation points and optional elements.

The UML can be easily extended in standardized ways. Therefore, even extensions can be standardized in form of a UML profile.

With variation points and optional elements generic models in UML are developed. Thus they use all advantages of UML, e.g. tool support and standardization.

Keywords: product lines, variability, software product families, generic models, feature models, UML,UML extensions

1 Introduction

The modeling of software is very necessary for the development of large and complex systems. Software models are abstractions from code, can serve as input for program generators and provide documentation to developers as well. The Unified Modeling Language (UML, [1]) is a standard modeling notation for software.

Modeling commonalities and variabilities is a key concept in development for reuse. It helps identifying and systematically developing common parts for reusability and is used in approaches for product lines and system families. The most common model notation is that of feature diagrams. These model features as an abstract description of the characteristics of a group of systems and focus on the end-user related properties of systems.

To realize such abstract product descriptions a so-called product line architecture is build that represent a generic model. This generic model can be instantiated for each product using generators or building individual products on top of a platform. The variability in such a model is modeled using variation points (hot spots) that identify the location and other attributes of differences between several products.

This paper is based upon a diploma thesis about modeling variabilities in UML [11]. The thesis conducts mostly all of the well-known modeling approaches (not completely listed in the references, see [11]) for variability and develops a standard-conform extension to UML (version 1.4) for modeling features and variability without violating the objectives of UML. Therefore the extensions are fully compliant with the specification and can serve as basis for proposing a standardized UML extension (called profile).

This paper focuses on the application of the variability extensions for generic modeling in UML. A more specific description of the feature modeling extension can be found in [12].

2 Generic Models in UML

Actually the UML is designed to model single software systems. But it defines mechanisms for extensions that can be formally specified and these can be used for extending the UML to describe generic models.

Generic models are used in domain engineering to describe product line architectures or to develop system families. Such models contain explicitly modeled variability and are instantiated for every product where the variabilities are bound. Therefore, variability describes the differences between instances of the system modeled and needs to be bound at specific points of time. The best known concept for modeling variability originates from the field of product family engineering: variation points [2]. For special modeling cases a more specific form of variation points can be identified: optional elements — the simplest form of a variation point.

These generic models are just like normal models, but contain explicitly modeled variability at so-called ‘hot spots’. These hot spots are build of variation points or optional elements and describe the generality in the model — usually on a relatively high level of abstraction.

From a bottom-up viewpoint the UML is already used in some modeling tools (e.g. Together) to develop program code (almost skeletons) from models and vice versa (reengineering). That indicates that it is possible to generate code from UML models. For this reason UML models including variability described by variation points and optional elements can be regarded as generic models that are instantiated when the code is generated from the model. Consequently the extensions developed could also be used to build such generic models containing variability on a relative low level of abstraction. It seems to be necessary to provide specific extensions for controlling program generators but these can

be realized simply with tagged values storing specific control settings.

Scalability problems of the resulting models should be resolved with tool support by simply hiding less-important information (tagged values) of the model. This produces specific views of an extensive model. The perceivability of UML models containing the extensions described now is also improved using the scalability mechanisms described in UML [1], e.g. viewing classes only with their name compartment.

2.1 Variation Points

A variation point (Vp) locates a variability and its bindings by describing several variants. Every variant is one way to realize that variability and bind it in a concrete way. The variation point itself locates the insertion point for the variants and determines the characteristics (attributes) of the variability.

For the extension we distinct between three parts of a variation point: the location of the variability (named variation point), several variants, and a relationship between every variant and the variation point. The relationship assigns every variant clearly to one variation point. The variation point and variants are decorated with stereotypes, «variationPoint» respective «variant».

Due to some limitations by the UML metamodel we apply the stereotype «variationPoint» to the model element containing the variability (in opposite to [3]). Thus the model will not contain additional virtual model elements and can be directly used to generate code fragments without special treatment.

Actually «variationPoint» and «variant» are specified for the metaclass `GeneralizableElement`. Thus the notion of variation points can be applied on classes, components, packages, collaborations and associations. With this baseClass it is also ensured that at least generalizations can be used to connect variants to the variation point (respective to implement it).

For further specification, «variationPoint» implies some tagged values determing the binding time and multiplicity of variants. The latter determines how many variants can be bound at binding time. The usage of each variant can be formally specified in a condition that determines when the variant is to be included (in derived systems).

In addition to the attributes there may be interactions between variants and to other model parts. They are modeled using dependencies and distinguished into mutual exclusion (‘mutex’) and ‘requires’.

To document evolutionary constraints between elements, dependencies stereotyped «evolution» can describe them. This is intended especially to support the evolution of the model and the system [4]. A more sophisticated description of such interactions can be found in [10].

Using the extension for variation points the assignment of each variant to the Vp is modeled by the UML notation for the used implementation technique (e.g. generalization, parameterization). As the implementation technique is not known from the beginning of

the modeling process, dependencies are used alternatively. In short: use dependencies if the implementation technique is not yet determined, or the appropriate UML modeling construct otherwise.

2.1.1 Multiple variation points

Assuming the example of a variation point in a single sorting component. It can be recognized that there may be more than one variation, e.g. the sorting algorithm and the sorted data types vary independently from each other. Thus it is necessary to be able to model more than one variation point per model element.

To gain this ability the variation point notation is extended to support multiple variation points. This adaptation is only needed if there is more than one variation point — otherwise the simplified notation can be used.

To distinguish them from each other every variation point needs a unique name that is also useful for documentation. This name is used to unequivocally assign a variant to its variation point. Technically this is realized by a tagged value in the variant that contains the index of the concerned variation points name (also documented in a tagged value).

2.1.2 Graphical notation of variation points in UML

The usage of this extension is shown in figure 1. It models two variation points in a generic sorting component. Variation point `0` varies the sorting algorithm and distinguishes three variants. As the implementation technique is not determined the variants are connected by dependencies to the Vp. Since `sort` contains more than one variation point, all variants must contain the tag `VariationPoint`.

The second variation point (index `1`) varies the type of the data to sort and is realized using parameterization. Therefore the variation points model element is modeled as a template class and the variants are assigned to the Vp by template bindings.

Variant `quick_sort` of variation point `0` also demonstrates the usage of a condition for variant selection narrowing the variation space (see subsection 2.3 for further explanation).

A real-world example for the usage of the extension is described in section 2.3. There the variation point is located in `PaymentInfo` and realized using generalization as implementation technique.

2.2 Optional Elements

Variation points always consist of at least two model elements (variation point and a single variant). For some purposes this is still not appropriate or can not be applied. In such cases optional elements can be used to specify model elements that are optionally included.

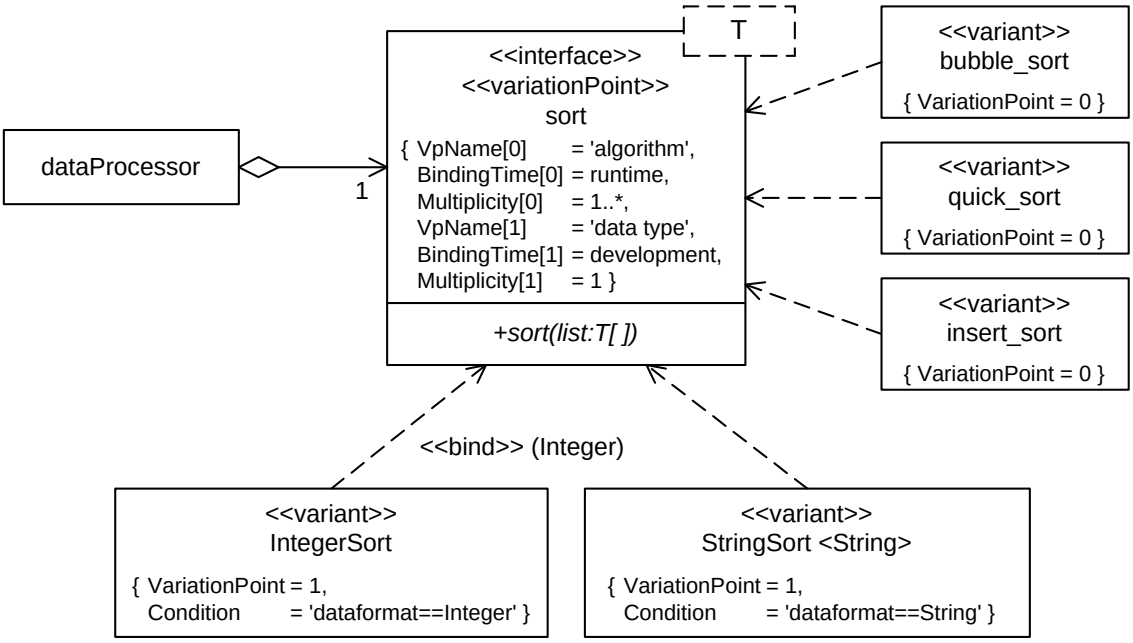

Figure 1: Example for multiple variation points

Their concrete inclusion can be specified with a condition analogous to variants; the time when this decision (of existence) will have to be taken at latest is described in the binding time. For the binding time five values are distinguished: at `development` time, during system `build`, at `installation` time, at system `startup`, or at `runtime`.

An example for the notation of optional elements can be found in figure 2 below.

2.3 An example for a generic model in UML

The sample model shown in figure 2 shows the usage of variation points and optional elements. The example has been taken from [11], where a piece of the eCommerce domain – the order process – has been modeled. The diagram models a part of the data structures to implement a simple order processing system.

It describes a single order consisting of several order items. Each item has an assigned price calculated with a specific pricing method. For instance, list prices are taken from the product information. The overall system optionally features contract-based pricing with discount based or flat prices. This feature ‘contract-based pricing’ is only included in highly qualified products and therefore marked as optional. As a result all system parts concerning this feature can be removed at compilation of the system leading to more compact binary code and less memory requirements.

The interdependency between the data class for discount storing and the existence

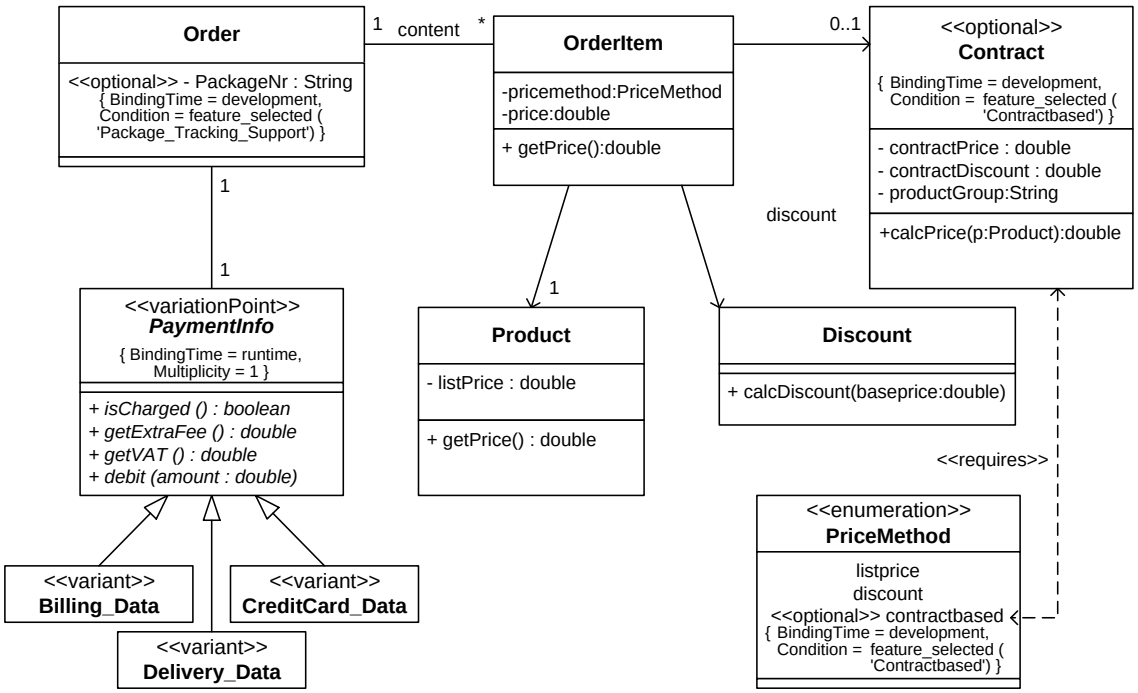

Figure 2: Sample data model for an order process

of the appropriate payment method is modeled by the dependency «requires» between `Contract` and the enumeration element `contractbased` of `PriceMethod`.

Another feature dealing with the tracking of delivery is realized in the attribute `PackageNr` of an order and only included if this feature is supported by the application. The used function `feature_selected()` serves as a reference to a higher-ranking feature diagram determing the existence of features.

The payment status of an order is stored via `PaymentInfo`. It is identified as a variation point with distinct behavior and data for every payment method. Thus every payment method is a variant and realized in a class that is a specialization of the `PaymentInfo` class.

3 Integrating Feature Modeling

For a systematic development for reusability the first step should be the explicit recognition of the features of the target system. These should be modeled in a feature model with an accentuation on the variabilities as known from domain engineering approaches [5, 7]. For generic models the feature model organizes the implemented variability as proposed in [6]. It also enables the modeling of configurations in UML but this is beyond the scope

of this paper.

Using the extension feature models can be described in UML. This notation supports all known kinds of diagram notations and additionally enables the use of UML typical concepts, e.g. constraints and the OCL. Further information on modeling features with UML can be found in [12].

For generic models it provides a way for abstract modeling of dependencies between several system parts [10]. Caused by the nature of features it also can model non-functional requirements, e.g. aspects or implementation techniques.

Features should be used to organize the variability expressed in the model. This can be realized using conditions in variable elements and is documented using UML abstractions (e.g. «trace» for traceability). There will be some more research on configuration modeling in UML and its application on generic models.

4 Usage of these extensions

Using variation points and optional elements generic models are described in UML. The extensions explicitly document variability in a system and provide a way to systematically develop with reuse.

It also includes possibilities to develop generators that produce system parts from generic UML models. There may be a need for more detailed specifications of variability to control such generators. These can be easily realized with the extension mechanisms of UML, namely stereotypes and tagged values.

One necessary step towards this will be the need for an integrated configuration modeling. There are some ideas but first the general usability and acceptance of the extensions needs to be verified.

5 Summary

The overall aim is to develop a standardized extension to UML that unifies the several existing notations and enables a broader awareness and usability of development with product lines/families.

Actually it provide a consistent, well-specified and uniform modeling notation that can be used for almost all known product line approaches and is extensible for further special purposes. It also enables the usage of the advantages of UML for system family development and provides a consistent notation for all modeling steps.

References

[1] Object Management Group (OMG), *OMG Unified Modelling Language Specification*, versions 1.3 and 1.4 Draft, March 2000 / February 2001

[2] I. Jacobson, M. Griss, P. Jonsson, *Software Reuse – Architecture, Process and Organization for Business Success*, Addison-Wesley Longman, 1997

[3] M. Coriat, J. Jourdan, F. Boisbourdin, *The SPLIT method*, In Proceedings of the First International Software Product-Line Conference (SPLC-1), August 2000

[4] 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

[5] K. Kang et al, *Feature-Oriented Domain Analysis (FODA) Feasibility Study*, Technical Report No. CMU/SEI-90-TR-2, November 1990, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA

[6] M. Griss, J. Favaro, M. d'Alessandro, *Integrating Feature Modelling with the RSEB*, International Conference on Software Reuse, June 1998

[7] K. Czarnecki, U. Eisenecker, *Generative Programming – Methods, Tools and Applications*, Addison-Wesley, 2000

[8] K. Kang, et. al., *FORM: A Feature-Oriented Reuse Method with Domain-Specific Reference Architectures*, In Annals of Software Engineering 5, 1998

[9] M. Svahnberg, J. van Gurp, J. Bosch, *On the notion of Variability in Software Product Lines*

[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, *Untersuchung der Modellierung von Variabilität in UML*, diploma thesis, August 2001

[12] M. Clauß, *Modeling variability with UML*, GCSE 2001 - Young Researchers Workshop, September 2001

The documents described in [10, 11, 12] can be found at:
http://www-st.inf.tu-dresden.de/~mc3/varUML/