A UML 2 Profile for Variability Models and their Dependency to Business Processes*

Birgit Korherr and Beate List
Women's Postgraduate College for Internet Technologies
Institute of Software Technology and Interactive Systems
Vienna University of Technology
{korherr,list}@wit.tuwien.ac.at
http://wit.tuwien.ac.at

Abstract

Variability Models are designed for modelling variabilities of a software. Unfortunately they are not part of a well-known modelling framework for a higher usability, like the Unified Modelling Language. To address this limitation, we provide a UML 2 profile for variability models. Furthermore we show the dependency from the UML profile to activity diagrams to make the relationship between variability models and process models visible. This profile and its mapping are tested with example business processes.

1 Introduction

Variability models define the variability of a software product line and can be used during the different life cycle stages of software product lines [7]. Variability modelling is a domain specific modelling technique, that is becoming more and more integrated into traditional software engineering. Unfortunately, it is not integrated into an modelling framework like the Unified Modeling Language (UML). Furthermore variability models have also an impact on processes. Variabilities can change the process flow, e.g. in a car engine manufacturing process the decision if the variability manufacture a diesel engine or a petrol engine is chosen, changes the process flow. The goals of this paper are to:

- provide variability models to software developers in a UML notation as well as through UML tools
- show the dependency between variability models and business processes to make the relationship between structural models and behavioural models in all stages of the software developing process visible.

UML profiles are an extension mechanism for building UML models for particular domains or purposes [6]. We utilise this well-defined way to develop a UML profile for variability models and describe its dependencies onto activity diagrams, to show the impact of variabilities on the process flow and thus, provide the following contributions:

- The UML profile for variability models can be easily created, presented and edited with existing UML modelling tools, as almost all newer UML tools support UML profiles.
- The profile provides variability models to software developers in UML notation. Since software systems are very complex and require variabilities for defining their process logic e.g. by customisation, the profile represents variability requirements to software developers or process engineers in a formal and well-known modelling notation.
- The UML profile and its shown dependencies onto activity diagrams makes the relationship between variabilities and processes visible.

In the remainder of the paper, variability models are briefly discussed in section 2. The UML profile which provides the concepts to present variability models in UML will be explained in section 3. In section 4, we describe a set of OCL constraints of the UML 2 profile to indicate restrictions that belong to the profile. The description of the dependencies from the profile to activity diagrams will be explained in section 5 followed by an example business process that underlines the communalities of the UML profile for variability models and activity diagrams. We close with related work (section 7), and the conclusion (section 8).

*This research has been funded by the Austrian Federal Ministry for Education, Science, and Culture, and the European Social Fund (ESF) under grant 31.963/46-Vii/9/2002.

2 The Concepts of Variability Modelling

A variability model shows the different variation points and variants of a software product line. Variability models are based on a metamodel developed by Pohl et al. [7], that is not MOF-compliant, because it has association classes which are not part of MOF. We adapt the metamodel of Pohl et al. [7] by changing the association classes and its associations with normal binary relationships and classes to make it MOF-compliant according to Hitz et al.[3]. The MOF-compliant metamodel could be for instance easily integrated into UML, which is based on MOF. The adapted metamodel is shown in figure 1.

A variability model consists of the variabilities *variation point,* and *variant*, as well as the possible relationships between them. A variation point is a representation of a variable item of the real world or a variable property of such an item, and has a *variability dependency* relationship with at least one variant. A variant is a representation of a particular instance of a variation point, and can constrain it with a constraint dependency. Furthermore it has at least one variability dependency to a variation point. A variability dependency is an abstract class, and can be distinguished between optional or mandatory. If a variation point has a *mandatory* dependency to a variant, then the variant has to be chosen if the variation point is selected. An *optional* variability dependency indicates that none, one or more variants can be selected. Moreover, it can be refined by an *alternative choice*, which declares the range between a variation point and its variants. The *constraint dependency* distinguishes an *requires* and an *excludes* dependency between variants, variation points and variants to variation points. An *requires* constraint dependency indicates that a variability is dependent on another variability, and an *excludes* constraint dependency defines that a variability has to eliminate another variability if it is selected.

3 A UML Profile for Variability Models

Referring to section 2, variability modelling is a domain specific modelling technique, that has no MOF-compliant metamodel for integration for instance into another modelling framework. The Meta-Object-Facility (MOF) is the four-layered metamodelling architecture of the OMG [4]. If a modelling technique has a MOF-compliant metamodel, then a UML Profile can be easily created, because UML offers a possibility to extend and adapt its metamodel to a specific area of application with profiles. A profile can extend a metamodel or another profile [6] while preserving the syntax and semantic of existing UML elements. It adds stereotypes which extend existing classes. Furthermore it specifies constraints and tagged values. A stereotype is a model element defined by its name and by the base class(es) to which it is assigned. Base classes are usually metaclasses from the UML metamodel, for instance the metaclass "Class", but can also be stereotypes from another profile. In this profile the metaclasses from Classes are extended to describe the different stereotypes. Also a stereotype can have its own graphical notation.

3.1. The extended metamodel with variabilities

A UML 2 class diagram describes the structure of a system that needs to be designed. It shows the main static properties as well as the possible relationship among each other of such a system, though it has no possibility to show the variabilities of a system. Therefore a section of the class diagram will be used as metaclasses to define a UML 2 profile for Variability Models for introducing variability concepts in UML. Figure 2 illustrates a section of the UML metamodel for Classes and its extension with stereotypes for representing variation points, variants, and their relationships among each other. The grey rectangles are the highlighted stereotypes of the profile. Furthermore we use the same icons for presenting the UML profile graphically as the variability model according to Pohl et al. [7] does.

3.2. Description of stereotypes

In a variability model a ≪*variation point*≫ marks out the different set of options such a model has. The OMG has defined a class as "a set of objects that share the same specifications of features, constraints, and semantics. Its purpose is to specify a classification of objects and to specify the features that characterize the structure and behavior of those objects." [6]. Therefore, classes are the appropriate metaclasses for describing the stereotype ≪variation point≫ and its characteristics, because a variation point describes like a class common properties of instances. Furthermore a subclass is a child from another class in a generalization relationship. A subclass inherits the structure, relationship and behaviour of its superclass and may add to it. The stereotype ≪*variant*≫ is a representation of a particular entity, and so far an extension of a subclass.

The different variability dependencies of a variability model, namely ≪*mandatory*≫, ≪*optional*≫, and ≪*alternative choice*≫ extends the metaclasses generalisation and generalisationSet to describe these different types of stereotypes. A generalisation is defined as a taxonomic relationship between a more general classifier and a more specific classifier, for example from a class to its superclass [6]. Each instance of the specific classifier is also an indirect instance of the general classifier. Thus, the specific classifier inherits the features of the more general classifier. A generalisationSet defines a specific set of generali-

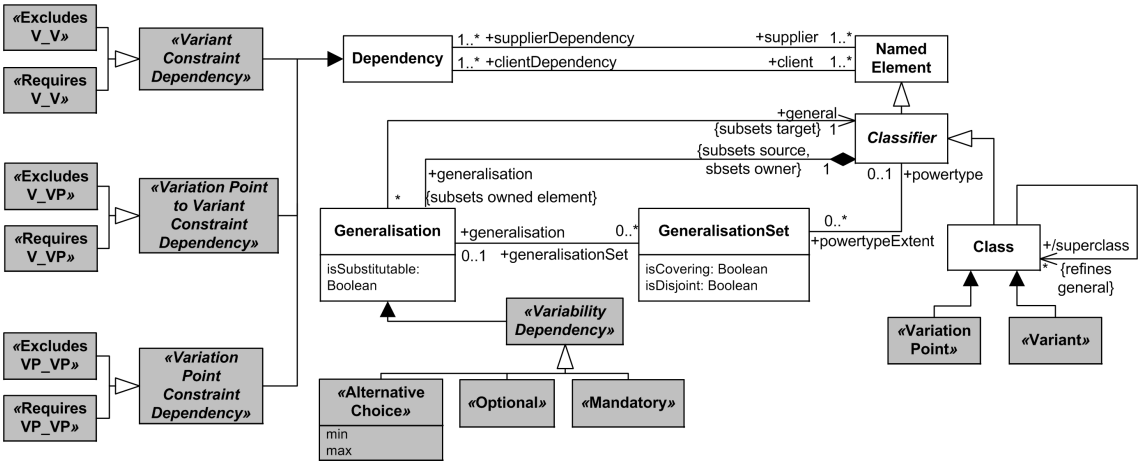


Figure 1. MOF-compliant metamodel of the variability model

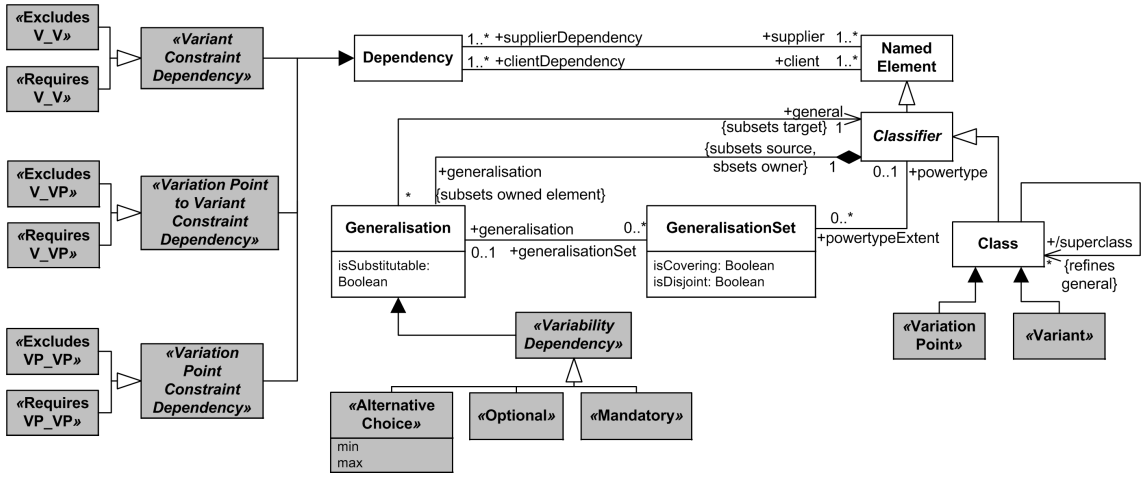
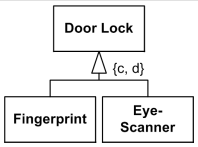
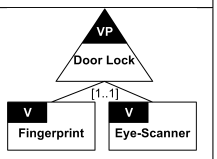
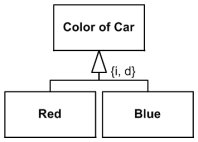
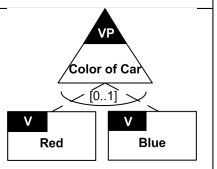
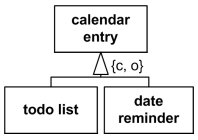
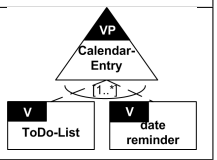
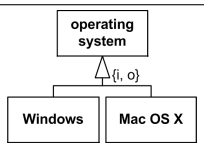
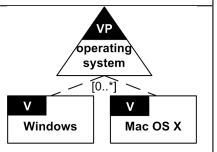


Figure 2. extended UML Class Metamodel with Variabilities

sation relationships. The metaclass describes how a general classifier (or superclass) may be divided using specific subtypes. Furthermore it has two metattributes with boolean values, namely isCovering and isDisjoint. If isCovering is true, then the generalisation set is complete, otherwise it is incomplete. On the other hand if isDisjoint is true, then the the generalisation set is disjoint, otherwise it is overlapping. Table 1 shows the different characteristics of variability dependencies with their multiplicities and how they could be described with the appropriate generalisationSet corresponding to the UML profile including examples to each couple. When isCovering is true, then every instance of an superclass has to be an instance of at least one of its subclasses at the same time. If isDisjoint is true, then every instance of a superclass corresponds only to one subclass. The overall multiplicity for the couple {complete, disjoint} is 1, that is the same like the multiplicity of the mandatory relationship, because it is defined that a variation point has to select the variant if it is part of the business process. For instance, the example in table 1 shows that a door lock can only be opened if a fingerprint and an eye-scan will be accomplished. But if isCovering is false, then a superclass can have more instances that do not correspond to the declared subclasses. This means the overall multiplicity for {incomplete, disjoint} is 0 to 1. This means that for example the color of a car can be either red or blue, or can have a different color. The alternative choice corresponds to that, because in such a relationship a group of variants can but do not need to be a part of the business process. If isDisjoint is false, then an instance of a superclass can have more then one related subclasses. The combination {complete, overlapping} is on a par with the alternative choice with the multiplicity 1 to *, which means that a variation point has to select at least one variant. On the one hand side a calendar entry can be a todo list, a date reminder or both.On the other hand side, the optional variability dependency defines that a variation point can select none, one or more variants, that is the multiplicity 0..*, the same like for the generalisation

Table 1. variability dependency and generalisation set

var. dependency	mult.	generalisation set	Class Diagram	UML Profile
Mandatory	1	{complete, disjoint}		
Alternative	0..1	{incomplete, disjoint}		
Alternative	1..*	{complete, overlapping}		
Optional	0..*	{incomplete, overlapping}		

couple {complete, overlapping}. An example would be that an operating system on a computer can be Win XP or Mac OS X, both of them, or a different one.

The variability constraint dependencies ≪*requires*≫ and ≪*excludes*≫ between the stereotypes of ≪variation point≫, ≪variation point≫ and ≪variant≫, as well as stereotypes of ≪variant≫ are defined by the metaclass dependency. With a dependency it is possible to show that an element, called client, is dependent on another element, called supplier. Dependencies are shown as dashed arrows. The model element at the tail of the arrow (the client) depends on the model element at the arrowhead (the supplier). The arrow may be labelled with an optional stereotype and an optional name. The ≪requires≫ stereotype defines that a client needs the consideration of a supplier. The ≪excludes≫ stereotype declares that a client excepts the consideration of a supplier.

4 Constraints

Constraints are applied to stereotypes in order to indicate restrictions. Constraints can be expressed in any language, such as programming languages or natural language. We use the Object Constraint Language (OCL) [5] in our profile, as it is more precise than natural language or pseudocode, and widely used in UML profiles. Tagged values are additional meta-attributes assigned to a stereotype, specified as name-value pairs. They have a name and a type and can be used to attach arbitrary information to model elements. Table 2 shows the stereotypes of the profile with its base classes, tagged values and constraints.

5 Mapping the Profile to Activity Diagrams

Variability models show the different variabilities of a software. Activity Diagrams are a part of the behavioural set of UML 2 diagrams, and are used for modelling business processes as well as for describing control flows in software.

On the one hand side variability models show the different variabilities of a software. On the other hand side UML 2 Activity Diagrams show the control and data flow between different tasks. The outcome of this is, that the two modelling techniques describe the same business case but the variability model describes the structural view and the activity diagram the bahavioural view. What we see here is that we need a mapping between these metamodels to examine in which way they are related to each other. Table 3 shows the mapped elements, with the variability metamodel on the left and the activity metamodel on the right.

A variant is fits to an action, because both describe an atomic task. Owing to that an activity partition is a kind of activity group for identifying actions that have some characteristic in common, it is mapped to a variation point, because it describes common properties of variants.

The optional variability dependency defines that a variation point can select none, one or more variants. Therefore this variability dependency is related to a fork node which decomposes one incoming flow in several concurrent outcoming flows and a join node, which synchronizes the several incoming flows. With additional guards on the flows it can be chosen if none, one, or more paths will be selected. The problem is that an activity diagram has no mechanism to integrate a default path, which would be necessary if no

Table 2. Stereotype Definitions of the UML Profile for Variability Models

Name	**Variation Point**
Base class	Class
Description	A variation point is a representation of a variable item of the real world.

Constraints	A variation point must have a superclass as a baseclass which is a role of a class.
	`context Variation Point inv:`
	`class.superclass=true implies variation point`

Name	**Variant**
Base class	Class
Description	A Variant is a representation of a particular instance of the stereotype variation point.

Constraints	A variant must not have a superclass as a baseclass which is a role of a class.
	`context Variant inv:`
	`class.superclass=false implies variant`

Name	**Mandatory**
Base class	GeneralisationSet
Description	A variant must be selected for an business process if and only if the associated variation point is part of the business process.

Constraints	If the variability dependency is mandatory, then metaattributes isCovering and isDisjoint of the metaclass GeneralisationSet are true.
	`context Variability Dependency inv:`
	`if self.Mandatory=true then`
	`self.GeneralisationSet.isCovering=true and`
	`GeneralisationSet.isDisjoint=true`

Name	**Optional**
Base class	GeneralisationSet
Description	A variant can but does not need to be part of a business process.

Constraints	If the variability dependency is optional, then metaattributes isCovering and isDisjoint of the metaclass GeneralisationSet are false.
	`context Variability Dependency inv:`
	`if self.Optional=true then`
	`self.GeneralisationSet.isCovering=false and isDisjoint=false`

Name	**Alternative Choice**
Base class	GeneralisationSet
Description	The alternative choice groups a set of variants that are related through an optional variability dependency to the same variation point and defines the range for the amount of optional variants to be selected for this group.
Tagged Values	min, max Type: UML::Datatypes::Integer Multiplicity: 1 Description: Min and max stands for the range of a relationship between variation points and variants.

Constraints	If the variability dependency is an Alternative Choice and the range is or smaller then 1, then the metaattribute isCovering is false and isDisjoint is true of the metaclass GeneralisationSet. Else if the tagged values of the alternative choice are min = 1 and max = n, then then the metaattribute isCovering is true and isDisjoint is false of the metaclass GeneralisationSet.
	`context Variability Dependency inv:`
	`if self.Alternative Choice=true and size()=<1 then`
	`self.GeneralisationSet.isCovering=false and isDisjoint=true`
	`else if self.AlternativeChoice=true and`
	`self.AlternativeChoice.min=1 and self.Alternative.max=n then`
	`self.GeneralisationSet.isCovering=true and`
	`GeneralisationSet.isDisjoint=false`

Name	**Requires**
Base class	Dependency
Description	The selection of a variability requires the selection of another variability.
Name	**Excludes**
Base class	Dependency
Description	The selection of a variability excludes the consideration of another variability.

path will be selected, because in that case the business process is terminated. To overcome this gap, it is necessary that the user defines an additional control flow, for instance a guard condition with an else statement.

The mapping of the alternative choice to an appropriate element of the activity diagram depends on the range of the variability dependency. If the range is 1 to *, then the alternative choice is mapped to a fork node to manage to concurrent flows, and closed with a join node for sychronisation of the incoming flows. Additional guards on the flows specify if one or more pathes will be selected. If the range is 0 to 1, then the alternative choice is mapped to a decision node which specifies with guards that at most one outgoing edge will be selected. Furthermore a merge node brings together the alternative flows. The problem is similar to the mapping of the optional variability dependency to activity diagrams. Also for an alternative choice with the multiplicity 0 to 1 a default path is needed to unsure that the business process does not terminate.

The *requires* and the *excludes* constraint dependency are

Table 3. Mapping Table from Variability Models to UML Activity Diagrams

Left MM Class	*Right MM Class*
Variation Point	Activity Partition
Variant	Action
Mandatory	Fork Node - Join Node
Optional	Fork Node - Join Node
Alternative Choice [1..*]	Fork Node - Join Node
Alternative Choice [0..1]	Decision Node - Merge Node
requires	Control Flow
excludes	Control Flow - Decision Node

both mapped to the control flow, independent whether if the dependency is between variants, variation points, or variants and variation points. In any case these elements are mapped as described above. Furthermore the *excludes* constraint dependency needs a decision node for its mapping to activity diagrams that the action that should be avoided cannot be executed.

6 Applying the UML Profile and its Mapping to an Example Business Process

We demonstrate the practical applicability of the UML profile for variability models and the dependency onto activity diagrams in figure 4 and 3. This small section of an example business process is part of the inventory process of a real international logistic company.

The variation point inventory accomplishment is distinguished between two types of inventory, *permanent inventory* and *periodical inventory*. An *inventory accomplishment* can be a *permanent* or a *periodical inventory* or both of them, when the two procedures overlap. Our example process is focused on the variant of the *periodical inventory*. It requires the variation point *base of inventory*, which has a mandatory relationship to the variant *inventory requirements*. It needs the variation points *inventory records*, *behaviour of logistics execution* and *generation of appointment* for further processing. For the *generation of appointment* the *number of positions* of the inventory goods can but need not to be used. The behaviour of the logistics execution depends whether a *printing device* or an `electronic device` or both of them are required. For further processing the variant *printing device* needs a special type of an *inventory record*, namely *list*, and the variant *electronic device* needs *radio*.

As the activity diagram example in figure 3 shows, every variation point of figure 4 has its corresponding activity partition, and every variant has its corresponding action. The whole business process is covered by the activity partition *inventory accomplishment*, which corresponds to the root

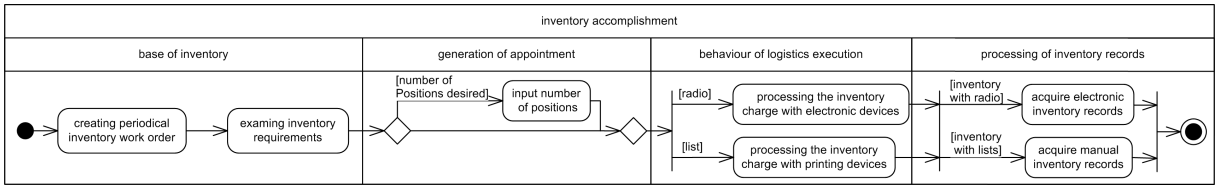


Figure 3. Example Business Process based on the Mapping

variation point in figure 4. The activity diagram starts with the action *creating periodical inventory work order*, followed by *examining inventory requirements*. Both are part of the activity partition *base of the inventory*, because the action *creating periodical inventory work order* requires the *base of the inventory*. If it is selected, then the `inventory requirements` continues with further processing. After that it can be chosen if the *input* of the *number of positions* of the inventory goods is desired or not, which is part of the *generation of appointment*. Afterwards the process continues with the decision if the *behaviour of logistics execution process* proceeds with the *processing the inventory charge with printing devices* or *electronic devices* or both of them. The process ends with the *processing of the inventory records*, distinguished if the *acquire of manual or electronic inventory records* or both of them is required.

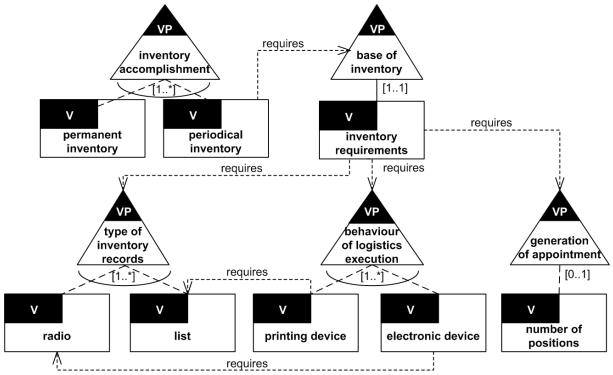


Figure 4. Example of a UML Profile for Variability Models

7 Related Work

The most relevant approaches in the global area of variability modelling will be presented now.

Rosemann et al. [8] proposed in their work configurable Event-Driven Process Chains (EPCs) as an extended reference modelling language. The difference to our approach is, that we do not want do describe the configurability of a certain business process modelling language. Furthermore we also consider the dependencies between structural and behavioural modelling techniques.

Clauss [2] introduces a UML extension to support feature diagrams which are an extension for the explicit representation of variation points. While Clauss integrates feature models in UML diagrams like the Use Case Diagram, our approach extends the UML metamodel to integrate variability models into UML.

Becker [1] developed a general metamodel for variability models on an examination of the most common concepts in variability modelling. This work describes variability models on a high-level. Our approach goes with the development of an UML-profile and the mapping to a business process modelling language more in detail.

8 Conclusion

We have presented a UML 2 profile for variability models to integrate the best concepts of variabilities and class diagrams in one model to overcome the gaps that variability models have no extension mechanism and no tool support. The UML profile for variability models can be easily created, presented and edited with almost all newer UML modelling tools. Moreover, we have shown the dependency between the UML profile and UML 2 activity diagrams, to make the relationship between structural models and behavioural models in all stages of the software developing process visible. The UML profile and its mapping were tested with an example business process.

References

[1] M. Becker. Towards a General Model of Variability in Product Families. In *First workshop on Software Variability Management*, 2003.

[2] M. Clauß. Modeling variability with uml. *GCSE 2001 Young Researchers Workshop*.

[3] M. Hitz, G. Kappel, and W. Retschitzegger. *UML @ Work*. dpunkt.verlag GmbH Heidelberg, 2005.

[4] OMG. *MOF 2.0 Specification*. http://www.omg.org, 2005.

[5] OMG. *OCL 2.0 Specification*. http://www.uml.org, 2005.

[6] OMG. *UML 2.0 Superstructure*. http://www.uml.org, 2005.

[7] K. Pohl, G. Böckle, and F. van der Linden. *Software Product Line Engineering*. Springer-Verlag Berlin Heidelberg, 2005.

[8] M. Rosemann and W. van der Aalst. A configurable reference modelling language. *Information Systems*, 32:1–23, 2007.

[9] T. Ziadi, L. Hélouët, and J.-M. Jézéquel. Towards a UML Profile for Software Product Lines. In *Software Product-Family Engineering, 5th International Workshop*, pages 129–139. Springer, 2003.