Design of a UML profile for feature diagrams and its tooling implementation

Thibaut Possompès, Christophe Dony, Marianne Huchard, Chouki Tibermacine

Design of a UML profile for feature diagrams and its tooling implementation

Thibaut Possompès*‡, Christophe Dony‡, Marianne Huchard‡, Chouki Tibermacine‡
*IBM France – PSSC Montpellier
Montpellier, France
thibaut.possompes@fr.ibm.com
‡LIRMM, CNRS and Montpellier 2 University
Montpellier, France
{possompes, dony, huchard, tibermacin}@lirmm.fr

***Abstract*—This paper proposes an instrumented solution to integrate feature diagrams with UML models to be used as part of a general approach for designing software product lines and for product generation. The contribution is implemented in IBM®Rational Software Architect (RSA). It is intended to be used in the context of large, complex and multi-domain projects, and at allowing model transformations to derive products. Our RSA implementation makes it possible to link feature diagrams with UML model artifacts. It allows traceability between feature models and other different kinds of models (requirements, class diagrams, sequence or activity diagrams, etc.). It is used in a project dedicated to create smart building optimization systems.**

I. Introduction

IT projects are closely related to business domains. However, there exist few techniques to gather, link and manage the knowledge related to each domain. Nowadays projects are particularly confronted to problems where instrumentation of specific business domains is required to enhance their efficiency. Examples of these projects are smart buildings, grids, water management, health care or food systems. The research presented in this paper is done in the context of the RIDER project[1], which brings together several actors who works on improving energy efficiency of buildings. This context requires to be able to link several domains, such as building system components, IT infrastructure, air flow modeling, building thermal management, database models, *etc.* Each domain can be modeled by a different stakeholder, or sometimes being based upon a standard. Software product line approach is perfectly appropriate to manage the variations that can be found in building instrumentation systems. The solutions presented in this paper are applied to the RIDER project and benefit of its feedback. Our approach is used as a part of a general approach for software product lines and for product generation.

There exist several commercial tools such as [3], [9], [21] that allow linking features to model artifacts extracted from various modeling tools; eclipse plug-ins [20], [1], [13], [10], [27], [15], [12]; and standalone feature modeling tools [22], [24], [26]. These tools do not implement the latest feature modeling concepts and are not well integrated in UML modeling tools. There also exist several UML profiles [5], [30] and meta-models [29] implementations but they address neither the latest UML specification nor all feature model concepts that are useful in our context.

This paper is organized as follows. Section II presents a new synthetic and expressive feature diagram model based on the state of the art. Section III presents our feature meta-model which synthesizes this state of the art. Section IV explains how the corresponding UML profile has been created and implemented using RSA, and shows an excerpt of the project feature model. Section V will sum up what has been presented and suggests some perspectives for further study.

II. Synthesis of Existing Feature Diagram Models

Various feature diagram semantics and implementations have been proposed since the initial one given in *FODA* [16]. We use the following criteria to analyze them and to define our model:
1 – Feature definition.
2 – Feature relationships. We identify four criteria to classify existing proposals regarding relations among features.

- *2.1 – Hierarchical relationships.*
- *2.2 – Feature choice constraint relationships.* necessary to guide the user through feature selection (feature dependency, or mutual exclusion).
- *2.3 – Mandatory and optional feature identification.*
- *2.4 – Sub-feature selection semantics.*

3 – Feature logical groups. Allowing to group arbitrary features by business domains, or abstraction layers.
4 – Product and implementation information. To determine what kind of information can help implementing a product from a set of selected features.

[1]The RIDER project ("Research for IT as a Driver of EneRgy efficiency") is led by a consortium of several companies and research laboratories, including IBM and the LIRMM laboratory, interested in improving building energy efficiency by instrumenting it.

Table I
HIERARCHY RELATIONSHIP

	Decomposition	Specialization	
		Enrichment	Realization
FODA [16]	Consists of		
FORM [18]	Composed of	Generalization Specialization	Implemented by
Fey et al. [11]	Refine		Provided by
Zhang et al. [29]	Decompose Detail	Specialize	
Czarnecki et al. [8]	Relation		

A. Feature Definitions

The initial definition of features was introduced by Kang et al. in *FODA* [16] which is extended by the *FORM* [17] method. Features are defined as being essential characteristics of applications, described with domain vocabulary.

FORM has introduced four different perspectives to enrich the semantics of feature diagrams: capability, operating environment, domain technology and implementation technique. Indeed, features are also able to model knowledge of the various domain experts involved in the project. Fey et al. [11] add the *pseudo feature* concept in order to allow for specialization with non-exclusive alternatives and to avoid feature redundancy thanks to implicit inheritance. Zhang et al. define features [28], [29] as being essentially a cohesive set of individual requirements representing the user-visible capability of a software system. Czarnecki et al. [8], [6] consider that features are system properties relevant for some stakeholder, and that any kind of functional or non-functional characteristic of a described system can be represented by a feature. This extends the *intention* definition presented by Zhang et al. [28].

The definition proposed by Czarnecki et al. allows more freedom of expressiveness in feature diagrams. Its combination with *FODA* perspectives allows us to apply feature diagrams to the numerous domains involved in a project. The *property* concept introduced by Fey et al. and the *attribute* concept presented by Czarnecki et al. are semantically very close. It could be assessed that a property expresses the same information that an attribute, *i.e.*, a measurable characteristic. We will use the *property* concept to describe specific feature typed values (*e.g.*, a room volume, *etc.*) and how one property can influence another one.

We extend this concept to allow the customer to choose the property value. Therefore, we will add the necessary semantics to restrict the possible choices to consistent ones.

B. Feature Relationships

1) Hierarchical Relationships: Features and sub-features can be bound by either decomposition or specialization. Table I illustrates the different variations on these concepts. The FODA *consists-of* relationship [4], [16], [25] is designed to represent decomposition. One parent feature can have several sub-features grouped either by an *AND* or *XOR* decomposition semantics. *FORM* [17], [18] introduces the relations *Composed-of*, to describe the constituents of a feature; *Generalization / Specialization*, to specialize or generalize a feature; and *Implemented-by*, to define how a high-level feature can be implemented by a lower-level one. Fey et al. [11] uses two kinds of hierarchical links between features: *refine*, to detail a given feature at a lower abstraction level; and *provided-by*, to link a pseudo-feature with the features which it realizes. *Pseudo-features* express an abstract functionality, quality or characteristic. Fey et al. make it possible to build directed acyclic graphs, which is impossible with *FODA*, by allowing one feature to refine several ones. Zhang et al. [29] identify three different kinds of hierarchy relationships: *decomposition*, to refine a feature into its constituents; *detailization*, to identify feature attributes; and *specialization*, to add further details into a feature. Czarnecki et al. [8], [6], [7] decided not to consider relationships between features in favor of entity-relationship or class diagrams.

We have chosen to keep the hierarchy relationships categorized in Table I. with the following concepts: *decomposition*, which consists in *detailing* the sub-features that compose the parent feature; *specialization*, which encompasses the concepts of *enrichment* for sub-features that add functions to the parent feature, and *Realization* (or *implementation*) that describes how a feature can be implemented.

2) Feature Choice Constraint Relations: FODA [16] uses the concept of *composition rules* to describe how features relate to one another: one feature can *require* another one, or two features can be *mutually exclusive*. Riebisch et al. [23] introduce the *hint* relationship which recommend features to the user.

We have kept the constraints *require* and *conflict* to ensure a coherent product generation. Furthermore the *hint* relation is also convenient to make recommendation to the user during the feature selection process.

3) Mandatory and Optional Features Identification: In the *FODA* [16] and *FORM* [18] specifications and tools, all features are mandatory by default; optionality is represented as a feature property as in [23], [14], [29]. Czarnecki et al. [6] use cardinalities to express optionality. For instance, a *(1,1)* cardinality describes a mandatory feature and *(0,1)* describes an optional one. Setting a cardinality greater than 1 specify how often the sub-features of a parent can be duplicated.

We keep for our model the cardinality based relationship introduced by Czarnecki et al. [6] that brings an interesting enhancement to the initial definition, useful to describe more complex products from a product line.

4) Sub-feature Selection Semantics: FODA [16] and *FORM* [18] methods propose two ways to manage sub-feature selection. A simple link between the parent and its sub-features means that there is no choice constraint; equivalent to an *or* binary choice. A semi circle drawn across the links means that there is an alternative choice; equivalent to the *xor* binary choice. Likewise, Fey et al. [11] express with a simple link that the choice is an *or*, but the alternative choice is expressed with a complete graph of mutually exclusive constraints among all sub-features. Riebisch et al. [23] and Czarnecki et al. [8] use cardinalities to define how many sub-features can be chosen.

The most convenient semantics relies on cardinality based selection semantics. It allows a maximum flexibility to describe how many sub-features can be selected. Furthermore, we chose to keep the *"or"*, *"and"*, and *"xor"* groups to ease the feature models semantics understanding for non-IT specialists end-users.

C. Feature Logical Groups

The *FORM* [18] method classifies features into four categories called layers, from a functional point of view. Riebisch et al. [23] use logical groups to represents aspects valuable to the customer and explain that *abstract features* could be used to encapsulate features related to a given concept. Fey et al. [11] present *feature-sets* to group features from an arbitrary point-of-view. Zhang et al. [29] present the *binding-time* concept to represent the phases of the software life-cycle in which each feature must be chosen. Czarnecki et al. [6], [8] use abstract features to reference other feature diagrams to reuse a set of features. They also introduce different types of features that can be considered as feature groups.

The *binding-time* concept can be extended according to the software development process chosen by the final user to allow him to assign features or groups of features to a development phase. It can be modeled by the *feature set* concept presented by Fey et al. The different kinds of features presented by Riebisch et al. [23] could be easily modeled by sub-layers. Hence, we choose to keep the layer concept to organize features in logical groups and sub-groups accordingly to the type of information they represent. We propose to enhance Fey et al. *feature-set* concept by adapting Zhang et al. [29] constraints meta-model to describe constraints inside a group of features, and constraints between two groups of features. The feature-sets could also be used along with the Kano method [19] to help the user choosing a set of product features that yield high customer satisfaction. The customer preference categories can be modeled as feature-sets encompassing the corresponding features. We also keep the feature-sets idea to reference sub-parts of a feature diagram. Hence, a feature-set could be a leaf of the diagram that encompasses another feature hierarchy. Staged configuration can be modelled by creating one feature-set for each configuration stage, and associating it with a group of stakeholders that have the same business concern. Czarnecki et al. [6], [8] have presented four types of features that have been integrated in our meta-model proposal: *concrete features* can be stored in the implementation layer; *aspectual features* can be stored in a sub layer of implementation layer; *abstract features*, *e.g.* performance requirements, can be represented by feature properties; and *grouping features* can be modeled as a feature set.

D. Product and Implementation Information

Riebisch et al. [23] argue that the feature hierarchy must be organized to make easier the choice of features by using composition relations and require associations. Zhang et al. [29] present a feature attribute for representing the feature *binding-state*. It must be used in a *binding-time* context, *i.e.* when features are implemented in the software product at a given software life-cycle phase. Mathematical relationships have been presented to describe the relative impact of one feature to another.

III. A Synthesis Meta Model for Feature Diagrams

This section describes a meta model synthesizing the choices we presented and motivated in the previous section. This work extends the work of Asikainen et al. [2] in order to apply our work in the context of a rich industrial project.

As depicted in Figure 1, a product line contains features, and a feature belongs to one product line. A product belongs to one product line and can be composed of as many features as needed. Features associated to a product must be analyzed in order to check whether all constraints, like mutual exclusion between features or require relations, are satisfied.

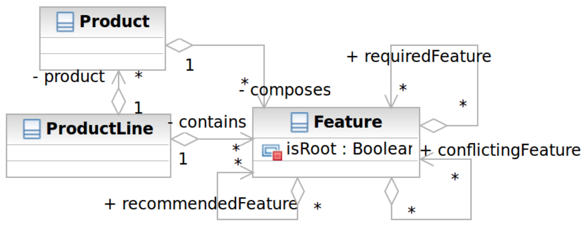

Figure 1. Product lines relation to features and products

Mutual exclusion and require relations link features regardless of their position in the hierarchy, they are modeled by the *conflictingFeature* and *requiredFeature* relationships. Furthermore, we add the *recommendedFeature* role used to advise a user to choose another feature pertinent in the product. These roles are depicted in Figure 1 in reflective association on the *Feature* class.

Feature properties (Figure 2) are used to describe either a feature parameter related to its inner requirements (*e.g.* the bandwidth capacity of a network) or a characteristic chosen by the user during the product definition (*e.g.* the frequency of automatic backups of a word processing software).

Features can have a variability type (*VariabilityKind*) which is further described below:

- *fixed*, A property value is fixed throughout all products of the product line.
- *variable*, A property value can change, within a product, depending on other features properties, *e.g.* the text buffer size of a text field in the user interface can vary accordingly to the type of information we want to store (*i.e.*, name or address).
- *family-variable*, A property can vary from product to product accordingly to the selected features, *e.g.* the phone book capacity depends on the presence of internal memory property and the sim card capacity.
- *user-defined*, A property value can be freely chosen in a given product, *e.g.*, the frequency of automated backups in a word processing software.

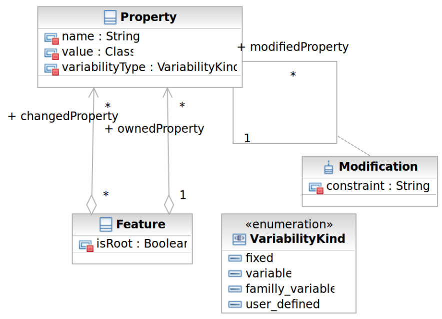

Figure 2. Feature properties

The hierarchy relationships and sub-feature groups are depicted in Figure 3. The relations between a feature and its sub-features are grouped by the *RelationshipGroup* class that contains the cardinalities necessary to restrict the number of sub-features to choose. Cardinalities can be chosen freely, but some groups have fixed cardinalities: *OrGroup*, *(0,*)*; *AndGroup*, *(*,*)*; and *XorGroup*, *(0,1)*. The *DirectedBinaryRelationship* class represents the kind of association that links a parent feature and a sub-feature. It is specialized either by *Enrich*, *Implement*, or *Detail* classes.

Figure 4 shows how layers and feature sets can be associated with the project stakeholders. A stakeholder represents any kind of people (*e.g.* domain experts, IT architects, *etc.*) allowed to choose features. Feature sets and layers can be attached to a concern specific to the project, *e.g.* network architecture, or business requirements. A *Layer* represents a specific view onto the software application, a feature can be in only one layer at a time. A *FeatureSet* inherits from the *Feature* class, and allows grouping features from an arbitrary point of view, *e.g.*, a business domain, or representing the features that must be implemented to fulfill a norm.

Figure 5 depicts how constraints can be applied to feature sets: *mutex*, when only one feature can be selected in the feature set; *None*, when there is no constraint among features; *All*, when all features or none of them can be selected. The *ConstraintRelation* class describes the relation between two feature sets: one feature set can require another one, two feature sets can mutually require each other, or be mutually exclusive. A *BindingPredicate* is used to represent how a constraint must be applied on each constrained feature-set. The choice of the specialized class must be made according to the kind of feature-set.

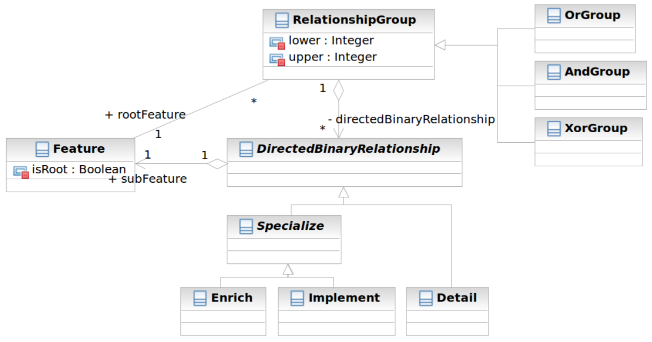

Figure 3. Groups and hierarchy relationships

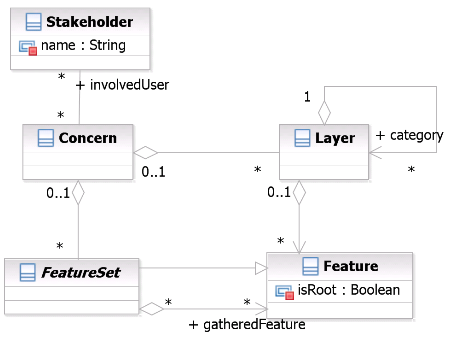

Figure 4. Stakeholders concerns

IV. UML Profile Implementation

A. Description

Our profile integrates the previously described feature meta-model into the UML meta-model hierarchy with an appropriate semantics. We describe here which UML meta-classes we have chosen to extend, which information has been added with the stereotypes and how we have restricted the semantics of extended meta-classes thanks to OCL constraints. However, the profile could be implemented in different ways by choosing to extend different meta-classes. We chose the meta-classes that had the closest semantics to our concepts, added the required information with the stereotypes and restricted the initial semantics of meta-classes to what is necessary for our profile with the Object Constraint Language (OCL) constraints.

Contrary to [5] we have based our feature diagram profile on the *Components* UML meta-class. A component, as a

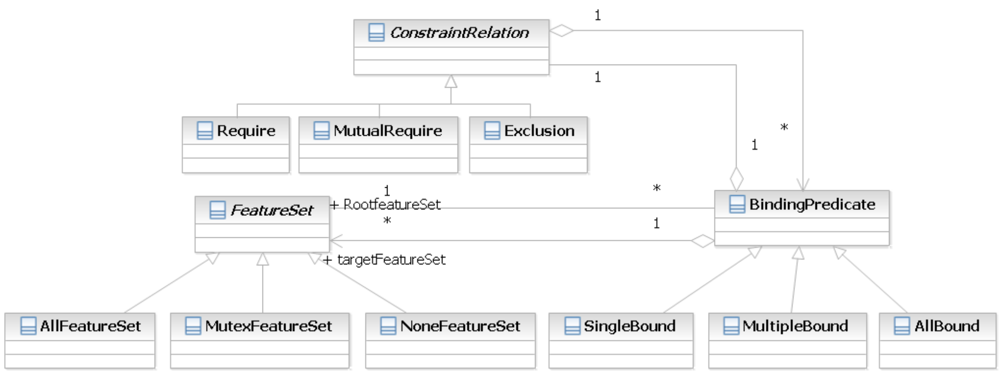

Figure 5. Constraints on feature sets

feature, can be seen as a high level view of a software element and, as such, is the concept the closest to what we want to express. Components have ports, which can be linked together. Ports are thus reasonably well suited to support the relationships between features and other elements defined in section III. It is the UML 2 concept the closest to what we want to express because it is a high level view of the structure of a software element. This first choice influences the other meta-classes selection.

Table II
STEREOTYPES EXTENSIONS

Stereotype	Extended Meta-class
Feature	Component
Stakeholder	Actor
Concern	Class
ModelRelationship	Dependency
Layer	Package
ProductLine	Component
Product	Component
Property	Port
RelationshipGroup	Port
Modification	Usage
DirectedBinaryRelationship	Association
BindingPredicate	Port
ConstraintRelation	Association

The *port* meta-class allows us to represent the interaction of a feature with other elements. It can be linked to other ports or components. The modification of a property value can be modeled by textual or OCL constraints placed upon the relationship between two properties.

We choose to create a Rational Software Architect plug-in to leverage its UML modeler, modeling editors, views and tools. All managed models are instances of EMF models. Hence using Rational Software Architect allows simplifying tasks like creating a specific plug-in for integrating feature modeling capabilities into standard EMF-based UML models and diagrams. The plug-in is currently used in the RIDER project.

Table II gives an abstract of our specialization choices, but due to space constraints we did not explicit all our design choices.

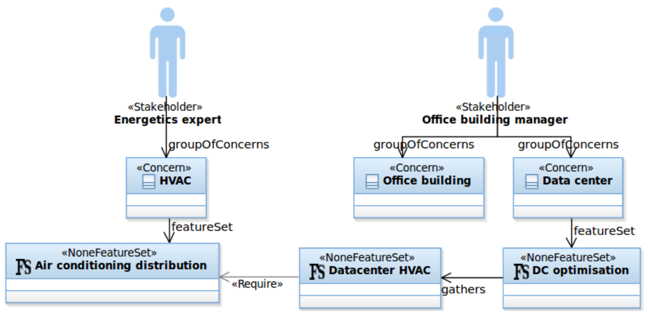

Figure 6. Stakeholders and feature sets

B. Example

Figure 6 shows an excerpt of a RIDER feature diagram expressed, using our profile, by two stakeholders having different concerns on a RIDER smart building project. One focusing on HVAC (Heating Ventilation Air Conditioning) domain and the second on data centers and office building optimization problems. Each feature set encompasses several features that are not shown in this example because of space limitation. The example shows in particular *require* high level constraints applying on groups of features (feature sets). The example also shows how *properties* can be used to add further information on the product context (such as the building volume).

The feature selection process is achieved by showing feature diagrams to the user accordingly to his concerns thanks to model transformations.

V. Conclusion and Perspectives

In this paper we have presented a new feature diagram model and its profile implementation in UML2. This feature diagram model is first based on a synthesis of existing ones, and it has then been improved to fit the requirements of the

RIDER smart buidings project in which it is applied. Thanks to Rational Software Architect, we have generated a tool making it possible to produce feature diagrams conforms to our model. This synthesis is achieved by classifying the existing concepts into categories. This approach allows a full integration of feature diagrams into UML models and facilitates model transformations. In comparison with [2] we add several concepts such as *layers*, *stakeholder concerns*, *feature-sets*, and *group constraints*.

For the time being, we still need to guide users to organize features into layers and sub-layers in order to best integrate them into the software development life-cycle. Hence, the next steps will be to create a framework able to use the full potential of our feature meta-model, and to develop automated model transformation functionalities to automatically generate UML models.

References

[1] M. Antkiewicz and K. Czarnecki. FeaturePlugin: feature modeling plug-in for eclipse. In *eclipse '04: Proceedings of the 2004 OOPSLA workshop on eclipse technology eXchange*, pages 67–72, New York, NY, USA, 2004. ACM.

[2] T. Asikainen, T. Mannisto, and T. Soininen. A unified conceptual foundation for feature modelling. In *Proceedings of SPLC '06*, pages 31–40. IEEE Computer Society, 2006.

[3] BigLever Gears. http://www.biglever.com/overview/software_product_lines.html.

[4] Y. Bontemps, P. Heymans, P. Y. Schobbens, and J. C. Trigaux. Semantics of FODA feature diagrams. In *Proceedings SPLC 2004 Workshop on Software Variability Management for Product Derivation*, pages 48–58, 2004.

[5] M. Clauss. *Untersuchung der Modellierung von Variabilität in UML*. Technische Universität Dresden, Diplomarbeit, 2001.

[6] K. Czarnecki, S. Helsen, and U. Eisenecker. Formalizing cardinality-based feature models and their staged configuration. *University of Waterloo*, 2004.

[7] K. Czarnecki, S. Helsen, and U. Eisenecker. Staged configuration using feature models. *Lecture notes in computer science*, 3154:266–283, 2004.

[8] K. Czarnecki, S. Helsen, and U. Eisenecker. Formalizing cardinality-based feature models and their specialization. *Software Process Improvement and Practice*, 10(1):7–29, 2005.

[9] M. Eriksson, H. Morast, J. Börstler, and K. Borg. *The PLUSS toolkit: extending telelogic DOORS and IBM-rational rose to support product line use case modeling*. ACM, 2005.

[10] Feature-Oriented Software Development Research. http://fosd.de/fide/.

[11] D. Fey, R. Fajta, and A. Boros. Feature modeling: A Meta-Model to enhance usability and usefulness. In *Software Product Lines*, pages 198–216. Springer, 2002.

[12] fmp2rsm Plug-in . http://gp.uwaterloo.ca/fmp2rsm/index.html.

[13] Generative Software Development Lab. http://gsd.uwaterloo.ca.

[14] M. L. Griss, J. Favaro, and M. d'Alessandro. Integrating feature modeling with the RSEB. In *In Proceedings of the Fifth International Conference on Software Reuse*, pages 76–85, 1998.

[15] Hydra. http://caosd.lcc.uma.es/spl/hydra/index.htm.

[16] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson. Feature-Oriented domain analysis (FODA) feasibility study. Technical report, Carnegie-Mellon University Software Engineering Institute, Nov. 1990.

[17] K. C. Kang, S. Kim, J. Lee, K. Kim, E. Shin, and M. Huh. FORM: a feature-oriented reuse method with domain-specific reference architectures. *Annals of Software Engineering*, 5(1):143–168, 1998.

[18] K. C. Kang, J. Lee, and P. Donohoe. Feature-Oriented product line engineering. *IEEE Software*, 2002.

[19] N. Kano, N. Seraku, F. Takahashi, and S. Tsuji. Attractive quality and must-be quality. *Journal of the Japanese Society for Quality Control*, 1984.

[20] ProS Labs - Moskitt Feature Modeler. http://oomethod.dsic.upv.es/labs/index.php.

[21] Pure-Systems GmbH. http://www.pure-systems.com/.

[22] RequiLine. http://www-lufgi3.informatik.rwth-aachen.de/TOOLS/requiline.

[23] M. Riebisch. Towards a more precise definition of feature models. *Modelling Variability for Object-Oriented Product Lines*, pages 64–76, 2003.

[24] S2T2 - An SPL of SPL Techniques and Tools. http://download.lero.ie/spl/s2t2/.

[25] P. Schobbens, P. Heymans, J. Trigaux, and Y. Bontemps. Generic semantics of feature diagrams. *Computer Networks*, 51(2):456–479, Feb. 2007.

[26] SPLOT - Software Product Line Online Tools. http://www.splot-research.org/.

[27] XFeature. http://www.pnp-software.com/XFeature/.

[28] W. Zhang, H. Mei, and H. Zhao. Feature-driven requirement dependency analysis and high-level software design. *Requirements Engineering*, 11(3):205–220, 2006.

[29] W. Zhang, H. Zhao, and H. Mei. A propositional logic-based method for verification of feature models. *Lecture Notes in Computer Science*, 3308:115–130, 2004.

[30] T. Ziadi, L. Hélouët, and J. M. Jézéquel. Towards a UML profile for software product lines. *Lecture Notes in Computer Science*, pages 129–139, 2004.