Formal Definition of Feature Models to Support Software Product Line Evolutions

Huilin Ye[1] **and Wendy Zhang**[2]

[1]School of Electrical Engineering and Computer Science,
University of Newcastle, Callaghan, NSW 2308, Australia
Huilin.Ye@newcastle.edu.au

[2]Computer Science & Industrial Technology Department
Southeastern Louisiana University, Hammond, LA 70402, USA
wzhang@selu.edu

Abstract - *Feature models have been widely used in software product line based software engineering. The dependencies between the variants and variation points in a feature model have very strong implications on the product configurations. Usually these dependencies are represented informally and incomplete in existing feature modelling approaches. In this work we first further explore the complex dependencies existing in software product lines. And then we propose a formal specification using Z notation to specify the dependencies in a product line. The specification formally defines software product lines and specifies complex dependency constrains contained in product lines. A set of operation schemas that support product line evolutions have been developed. With these operation schemas the invariants defined in the formal specification of product lines can be ensured when new features and feature dependencies are added into or removed from the product line. As Z specifications provide proof mechanism to validate the formal model and natural transition from a specification to an implementation a reasoning mechanism and a feature modelling tool can be developed in future.*

Keywords: *product line evolution, feature modelling, feature dependency, formal specification, Z notation*.

1 Introduction

Feature models are suggested as a useful abstraction to represent variability in software product lines [1]. Features are prominent and distinctive system requirements or characteristics that are visible to various stakeholders in a product family [2]. Feature oriented modelling approaches have been widely used in software product line engineering. Most of the approaches use diagrams to represent features and their relationships. Although the diagrams provide intuitive pictures of feature models there is not sufficient accurate definitions of the concepts used in feature modelling. This brings ambiguous semantics of feature models. Formal specifications are believed an effective means that provides a theoretical foundation for the principles of software engineering in general. Some research works that applied formalism to feature modelling have been reported [3-6]. But we think that the following issues of the formalisation have not been well addressed in the previous works.

- Feature dependency modelling: the lack of complete and accurate of formal specifications of feature dependencies has great implication on member product configurations in software product lines. Dependencies in feature models not only exist between features but also exist between features and variation points and between variation points. Understanding these different types of dependencies will help configure valid member products.

- Support product line evolutions: Software product lines will constantly evolve along the time. It must be guaranteed the invariants of a product line remain unchanged when new features and dependencies are added in or existing features are removed from the product line.

- Feature modelling tools are necessary for product line engineering. Connection from the formal specification to the implementation of the modelling tools should be established. Formal approaches distant to tool implementation are not practicable and may not be acceptable by the software industry.

This work addressed the above issues. Thus it contains several contributions. First, dependency relationships in feature models will be further explored in addition to the ones reported in the literature. Then a formal definition of software product line will be presented. A set of operation schemas that support product line evolutions have been developed. With these operation schemas the invariants defined in the formal specification of product lines can be ensured when features and dependencies are added into or removed from the product line. The remainder of the paper will be organised as follows. Section 2 reviews the relevant background of feature modelling and further explores feature dependency relationships in feature models. Section 3 presents a formal definition of product line and a set of operation schemas

supporting product line evolution. Section 4 concludes the paper.

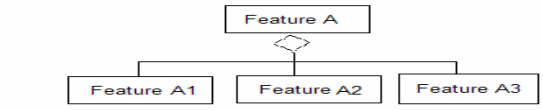
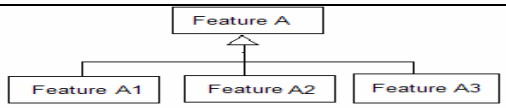
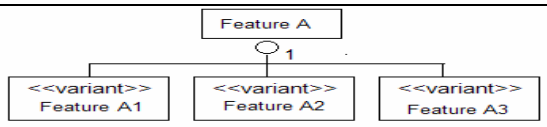
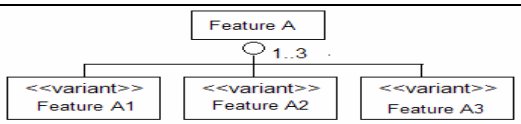
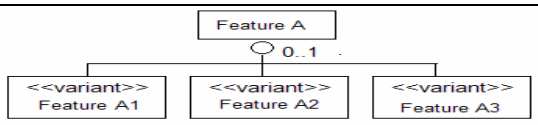
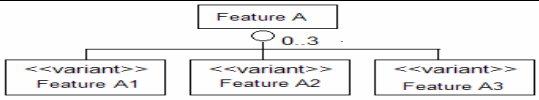
2 Background of Feature Modelling

2.1 Review of Feature Models

Features in a product line are classified as mandatory features and variable features to represent commonality and variability of the member products respectively. An important concept used in feature modelling is *variation point*. Currently existing feature modelling approaches usually use a tree structure to organize features. If a parent node of the tree has one or more variable child features, called variants, the parent feature together with variable child features is defined as a variation point [7]. The configuration of a member product in a product line will go through the feature tree to include all the mandatory features and select some of the variable features at each variation point. Variable features include alternative features, multiple alternative features, optional alternative features, optional multiple alternative features. Instead of using different symbols to represent these different types of variable features multiplicity is introduced to each variation point to differentiate the four kinds of variable features. Hierarchical feature relationships and different variability types are represented in Table 1 using extended UML notations. The semantics of the notations are also explained in the table.

Figure 1 shows a feature model for a `Car` product line [8]. This simplified model is used to demonstrate the abovementioned concepts rather than a real model for a car product line. In this feature model `Car` is composed by variation points `Control`, `Ordinary Accessories`, `Luxury accessories`, `Secure device`, `Quality attributes` and mandatory feature `Engine`. The otherwise features are variable features that can be identified by the attached stereotype `<<variant>>`. `Manual` and `Automatic` are the specialised features of `Control`. `Quality attributes` consists of `Security` and `Reliability`. `Control` is composed by `Manual` and `Automatic`. Its multiplicity is `1,` which specifies that its two variants are alternative variants, i.e. either `Manual` or `Automatic` can be chosen for a product configuration. `Ordinary accessories` has two optional variants represented by the multiplicity `0..2` attached to it. A car can have no such accessories, or can have either `fan` or `power steering`, or can have both. `Quality attributes` has multiple alternative variants represented by the multiplicity `1..2` attached to it. One or two quality attributes may be chosen when configuring a member product.

Table 1. Hierarchical Feature relationships and notations.

Relationships	Notations	Semantics
Composition		Feature A consists of Feature A1, A2, and A3
Generalisation/ Specialisation		Feature A is a generalised feature of Feature A1, A2, and A3
Variation point (Alternative)		One and only one feature can be chosen from {A1, A2, A3} at this variation point.
Variation point (Multiple alternative)		One or more features can be chosen from {A1, A2, A3} at this variation point.
Variation point (Optional alternative)		No feature or at most one feature can be chosen from {A1, A2, A3} at this variation point.
Variation point (Optional multiple alternative)		No feature or more features can be chosen from {A1, A2, A3} at this variation point.

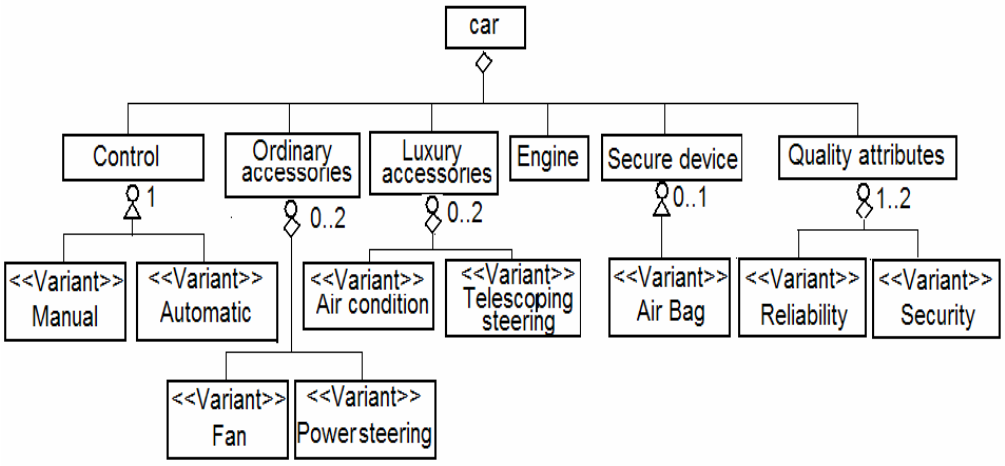


Figure 1. A simplified feature model for a `Car` product line.

2.2 Dependencies in Feature Models

Dependencies in a feature model specify the constraints on the selection of variable features when configuring member products. There are three kinds of dependencies, dependencies between two variable features, dependencies between a variable feature and a variation point, and dependencies between two variation points.

2.2.1. Dependencies between Variable Features

Three types of dependency relationships have been identified as follows:

1. `Requires`: If a feature requires, or uses, another feature to fulfil its task, there is a `Requires` relationship between the two features.
2. `Excludes`: If a feature conflicts with another feature, they cannot be chosen for the same product configuration, i.e. they mutually exclude each other. There is a bi-directional `Excludes` relationship between the two features.
3. `Impacts`: When a feature is selected for a certain configuration and the selection will have impact on another feature, it is called `impacts` relationship between the features.

Feature dependency relationships are non-hierarchical. We use UML stereo types to represent different types of dependencies (see Table 2). The dependencies identified for a product line must be validated. For a complex product line involving a large number of features, some conflicting dependencies may be recorded without awareness of their existence [9]. The validation of dependencies is intended to discover these conflicting dependencies. The following rules are defined for the validation:

- Excludes relationship must be mutually exclusive.
- Requires and Excludes cannot be occur between the same pair of features.

Table 2. Feature dependency relationships and notations.

Dependency type	Notations	Semantics
Requires	Feature A <<Requires>> Feature B	Feature A requires Feature B.
Excludes	Feature A <<Exculdes>> Feature B	Feature A excludes Feature B and Feature B excludes Feature A.
Impacts	Feature A <<Impacts>> Feature B	Selection of Feature A will have impact on Feature B

2.2.2. Dependencies between Variation Points

There are two types of variation points, mandatory and optional, that can be distinguished by the multiplicity associated with the variation points. A multiplicity of 0..* or 0..1 means optional variation point while the otherwise multiplicity types represent mandatory variation points. For a mandatory variation point at least one variant must be selected. For an optional variation point the variants associated with the variation point may or may not be selected. Selecting or not selecting variants from an optional variation point will generally be based on a certain product configuration requirement. However, dependencies between variation points may constrain this selection. Selection of some variants at one variation point may cause dependencies to the other variation points in a product line [10]. For example, selection of some variants from one variation point may require or exclude the selection of variants from another variation point irrespective of which variant is selected. If both variation points are mandatory variation points `Excludes` dependency is not permitted and `Requires` dependency has already supported. The only case needed to be considered is the dependencies between optional variation points. Assume that both variation point *A* and *B* are optional variation points. If *A* requires *B* when some variants are selected at *A* some variants at *B* must also be selected irrespective of which variant is selected from both variation points. If *A* excludes *B* when some variants are selected at *A* no variant at *B* can be selected or when some variants are selected at *B* no variant at *A* can be selected irrespective of which variant is selected from both variation points. In the car product line as depicted in Figure 1, both `Ordinary accessories` and `Luxury accessories` are optional variation points. There is an `Excludes` dependency between the two variation points. If `Fan` or/and `Power steering` is/are selected no variants can be selected at `Luxury accessories` or if `air condition` or/and `Telescoping steering` is/are selected no variants can be selected at `Ordinary accessories`.

2.2.3. Dependencies between a Variable Feature and a Variation Point

The selection of a variant from one variation point may require or exclude the selection of variants from another variation point without constraining on the selection of the variants. If the required or excluded variation point is a mandatory variation point it will not be a problem as `Excludes` dependency is not permitted and `Requires` dependency has already supported. However, if there is a `Requires` dependency between a variant and an optional variation point the selection of the variant will force the optional variation point to become a mandatory variation point, i.e. some variants must be selected from this variation point. If there is an `Excludes` dependency between a variant and an optional variation point the selection of the variant will prohibit any variants being selected from the optional variation point. In the car product line as depicted in Figure 1, there is a `Requires` dependency from variable feature `security` and the optional variation point, `secure device`. If `security` is selected, `air bag` must be selected at the optional variation point `secure device`.

3 Formal Specification for Product Line Evolution

Figure 2 shows an updated version of a formal definition of product line reported in [11]. A product line consists of a set of features, variation points, and three kinds of dependencies. The relation *features* maps the features existing in a product line to the two different types (mandatory and variable). The *variationpoints* specifies a collection of variation points. The relation *dependencies1* maps the dependency types to different sets of feature pairs. The relation *dependencies2* maps the dependency types to different sets pairs of feature and variation point. The relation *dependencies3* maps the dependency types to different sets of variation point pairs. The detailed explanation of the formal definition can be found in [11].

ProductLine

$$features: FeatureType \pfun \mathbb{P}\ FeatureInstance$$
$$variationpoints: \mathbb{P}\ VPInstance$$
$$dependencies1: DependentType \pfun \mathbb{P}\ (FeatureInstance \times FeatureInstance)$$
$$dependencies2: DependentType \pfun \mathbb{P}\ (FeatureInstance \times VPInstance)$$
$$dependencies3: DependentType \pfun \mathbb{P}\ (VPInstance \times VPInstance)$$

$$\forall ft1, ft2: FeatureType \mid ft1, ft2 \in dom\ (features) \land ft1 \neq ft2 \bullet$$
$$features\ ft1 \cap features\ ft2 = \varnothing$$
$$\forall vpins: VPInstance \mid vpins \in variationpoints \bullet$$
$$vpins.parent \in \operatorname{ran}(features) \land vpins.variants \in \operatorname{ran}(features)$$
$$\forall dt1, dt2: DependentType \mid dt1 = Requires \land dt2 = Excludes \bullet$$
$$(dependencies1\ dt1 \cap dependencies1\ dt2 = \varnothing \land$$
$$dependencies2\ dt1 \cap dependencies2\ dt2 = \varnothing \land$$
$$dependencies3\ dt1 \cap dependencies3\ dt2 = \varnothing)$$

Figure 2. *ProductLine* schema

When a product line is defined it will be constantly evolved. New features and dependencies will be introduced to and some old features and dependencies may be removed from the product line. The *AddFeature* operation schema (Figure 3) is defined for adding a new feature into a product line. If a feature being added, *NewFeature?,* is a variable feature it may be attached to an existing variation point or a new variation point will be created by the addition. If it is added to an existing variation point the multiplicity may need to be updated by the addition. If this addition will create a new variation point, a multiplicity must be specified. In either case, an input variable *Multiplicity?* is needed to specify the multiplicity.

The predicate part of the *AddFeature* asserts the following:

- If *NewFeature?*is not an existing feature in the product line, add it in.
- If *NewFeature?* is a variable feature and its parent is the parent of an existing variation point, include it to the *variant*s of this variation point.
- If *NewFeature?* is a variable feature and the parent of *NewFeature?* is not the parent of any existing variation points the addition of *NewFeature?* will create a new variation point. The *id* of the new created variation point will be the number of existing variation points plus one. The parent of the new created variation point is the parent of *NewFeature?*. Its multiplicity is *Multiplicity?*. Its variant is *NewFeature?*. The *AddFeature* uses a μ expression. The semantics of a μ expression is the same as that of a set comprehension except that a μ expression returns only one value for the set.

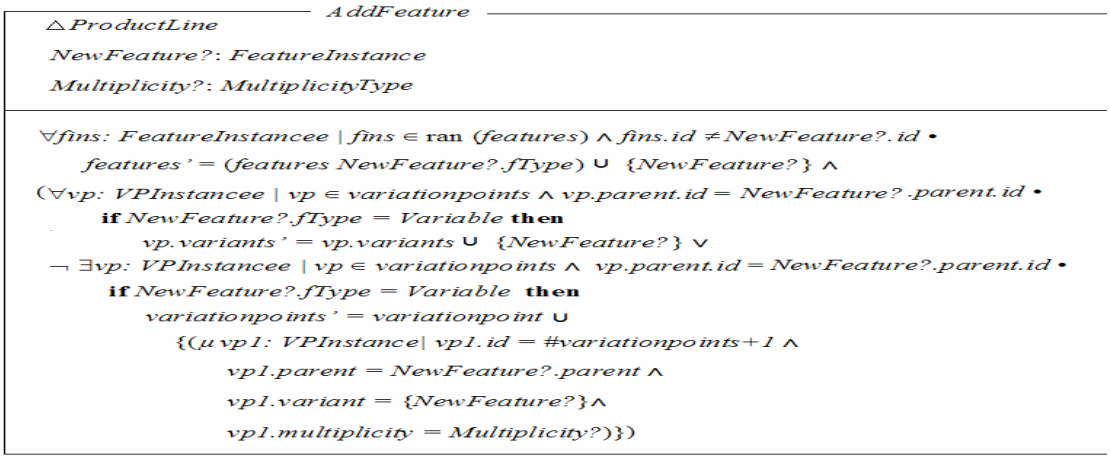


Figure 3. *AddFeature* schema.

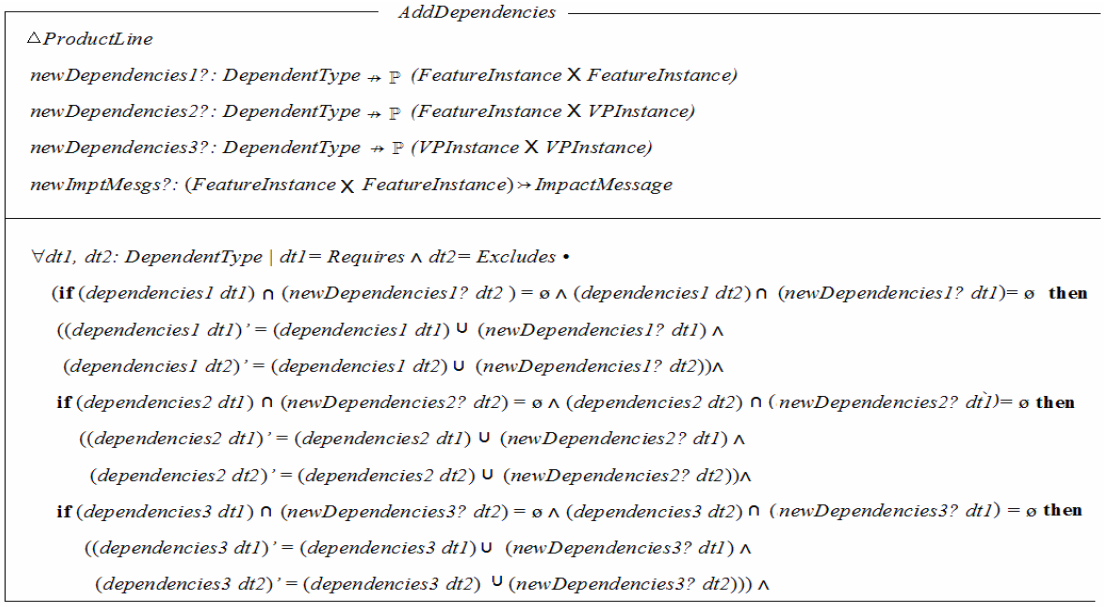


Figure 4. *AddDependencies* schema.

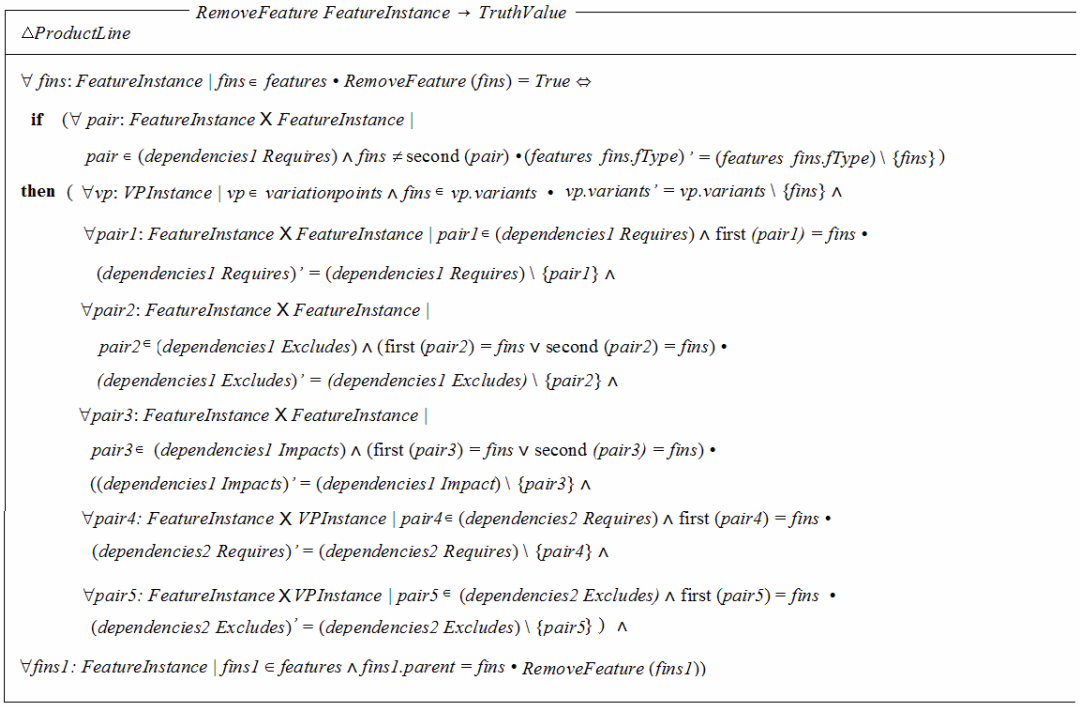


Figure 5. *Removefeature* function.

The addition of new features may introduce new dependencies into product lines. New dependencies can also be identified among existing features and variation points. The *AddDependencies* schema (Figure 4) is used to add dependencies to product lines. The relation *newDependencies1?* maps the new dependencies to be added between variable features to different dependency types. The relation *newDependencies2?* maps the new dependencies to be added between variable features and variation points to different dependency types. The relation *newDependencies3?* maps the new dependencies to be added between variation points to different dependency types. The relation *newImptMesgs?* maps a pair of features with `Impacts` relationship to an impact message.

The predicate part of the *AddDependencies* schema asserts that the new added set of dependencies should not conflict with the existing dependencies, i.e., a `Requires` and `Excludes` dependency should not exist between a pair of features, or between a feature and a variation points, or between a pair of variation points, at the same time in a product line.

A function named *RemoveFeature* (Figure 5) is used to remove a feature from a product line. The feature to be removed is given as a parameter of the function. The predicate part of the *RemoveFeature* asserts the following:

- If the feature to be removed is not a required feature by any of the other features in the product line, remove it. When the feature is deleted the following constraints must be satisfied.
- If the deleted feature is a variant of a variation point remove it from the set of variants of the variation point.
- Remove all the dependencies involving the deleted feature, including dependencies with other features and dependencies with optional variation points.
- *RemoveFeature* is a recursive function. If a deleted feature is a parental feature all of its child features should be recursively deleted from the product line.

4 Conclusions

The formal specification presented in this paper provides a generalised approach to product line engineering and has several advantages. It supports product line evolution. The defined operation schemas will detect conflicting dependencies introduced by adding new features into or by removing old features from a product line. It will be difficult for the logic expression based approaches to support product evolution as there is no facility to check any inconsistent dependency after a product line is updated. Based on the formal specification a set of theorems can be developed and proved. The reasoning mechanism and proofs of the formal specification will be our future work. Based on the formal specification a modelling tool for product line engineering can be easily developed as Z specifications provide natural transition from the specifications to an implementation. It will be much easier to implement a modelling tool based on the Z specification than any of the formal models using logic expressions. A modelling tool for product line engineering and product configuration will be developed in the near future.

5 References

[1] Gurp, J., Bosch, J., and Svahnberg, M., "On the Notion of Variability in Software Product Lines". In Proceedings of the Working IEEE/IFIP Conference on Software Architecture (WICSA'01), 45-54, 2001.

[2] Lee,, K., Kang, K., and Lee, J., "Concepts and Guidelines of Feature Modelling for Product Line Software Engineering". In Proceedings of 7th International Conference of Software Reuse, LNCS, Vol. 2319, 62-67, 2002.

[3] Batory, D., "Feature Models, Grammars, and Propositional Formulas". SPLC2005, Lecture Notes of Computer Science, Vol. 3714, 7-20, 2005.

[4] Benavides, D., Trinidad, P., and Ruiz-Cortés, A., "Automated Reasoning on Feature Models". In Proceedings of 17th International Conference on Advanced Information Systems Engineering, Lecture Notes of Computer Science, Vol. 3520, 2005.

[5] Mannion, M., "Using First-Order Logic for Product Line Model Validation". In Proceedings of 7th International Software Product Line Conference, Lecture Notes in Computer Science, Vol. 2379, pp. 176-187, 2002.

[6] Sun, J., Zhang, H., Li, Y., and Wang, H., "Formal Semantics and Verification for Feature Modelling". In Proceedings of the 10th IEEE International Conference on Engineering of Complex Computer Systems, 2005.

[7] Riebisch, M., BÖllert, K., Streitferdt, D., and Philippow, I., "Extending Feature Diagrams with UML Multiplicities". In Proceedings of the Sixth Conference on Integrated Design and Process Technology, Pasadena, CA, June 2002.

[8] Ye, H. and Liu, H., "An Approach to Modelling Feature Variability and Dependencies in Software Product Lines". IEE Proceedings – Software, Vol. 152, 101-109, 2005.

[9] Ye, H. and Sharmin, A., "Modelling Feature Variability and Dependency in Two Views". In Proceedings of the 17th International Conference on Software Engineering and Knowledge Engineering, Taipei, Taiwan, 661-664, July 2005.

[10] Bühne, S., Günter, H., and Pohl, K., "Modelling Dependencies between Variation Points in Use Case Diagrams". In Proceedings of 9th International Workshop on Requirement Engineering – Foundation for Software Quality, 59-70, June, 2003.

[11] Ye, H. and Lin, Y., "A Formal Specification for Product Configuration in Software Product Lines". In Proceedings of 19th International Conference on Software Engineering and Knowledge Engineering, July 2007.