Streamlined Feature Dependency Representation in Software Product Lines

H. Ye[1], Y. Lin[1], and W. Zhang[2]
[1]School of Electrical Engineering and Computer Science
The University of Newcastle, Callaghan, NSW 2308, Australia
[2]Computer Science & Industrial Technology Department
Southeastern Louisiana University, Hammond, LA 70402, USA

Abstract - *Feature dependencies have very strong implications on the configurations in a software product line. Different types of dependencies between features and variation points make dependency validation and product configuration very complex. An approach to streamlining dependency types is proposed to transform different types of dependencies to a single type of dependencies – dependencies between variable features without changing the configuration implications. The semantics of the transformed dependencies become much more intuitive, which makes the product configurations more effective and efficient. As the configuration constraint of a transformed dependency is simplified the dependency validations in feature models becomes much easier. A case study based on a Library Software Product Line has been presented to demonstrate how the proposed approach works.*

Keywords: Feature Dependency, Feature Modelling, Software Product lines

1 Introduction

Feature modelling has been widely used in product line based software engineering. Since the introduction to FODA [1], many extensions on feature models have been proposed. Most notable extensions include *variation point* (VP) and *feature cardinality*. Griss et al. [2] defined variation point in the context of feature models. A parent feature, together with the set child features decomposed from the parent feature by a variation decomposition operator, is called variation points. Czarnecki and Eisenecker [3] introduced group cardinality to feature models. A group cardinality takes the form of `k..m` and is associated with each variation point to specify between a minimum k features and a maximum *m* features that can be selected from a set of *n* variable child feature of the variation point, where 0<=k<=m<=n.

Dependency modelling is an important issue in feature modelling. Dependencies are the constraints on *product configurations* in software product lines. Product configuration is a process of selecting configurable features from a feature model for developing a product in a software product line. The following two dependencies have been identified by the previous works [1].

- `Requires`: If a feature requires, or uses, another feature to fulfil its task, there is a `Requires` relationship between the two features.
- `Excludes`: If a feature conflicts with another feature, i.e. they mutually exclude each other in a configuration. There is a bi-directional `Excludes` relationship between the two features.

The above two dependencies occur not only between features but also between variation points or between a feature and a variation point. Bühne et al. [4] discussed dependencies between variation points and dependencies between a feature and a variation point. The `Requires/Excludes` dependency between two variation points means selection of some variants at one variation point requires/excludes the selection of some variants from another variation point irrespective of which variants are selected from both variation points. Similarly, the `Requires/Excludes` dependency between a feature and a variation points means that selection of the feature requires/excludes the selection of some variants from the variation point irrespective of which variants are selected from the variation point.

Dependency modelling has great implications on product configurations in software product lines. The existence of dependencies between variation points and dependencies between a feature and a variation point make the dependency validations and product configurations in a feature model much more complex than that in a feature model where dependencies exist only between features.

In this paper we present an approach to transform dependencies between variation points and dependencies between a feature and a variation point to dependencies between features. This transformation will result in a streamlined feature dependency representation in feature models. Consequently dependency validations and

product configurations in such a feature model will become much easier.

The remainder of the paper will be organized as follows. Section 2 will define some basic concepts in feature modelling. The transformation approach will be presented in Section 3 based on the concepts defined in Section 2. A case study that demonstrates how the proposed approach works will be presented in Section 4. Section 5 will conclude the paper.

2 Feature and variation point

A feature model is a hierarchical tree structure consisting of three kinds of features, full mandatory features [5], mandatory features [6], and variable features [1].

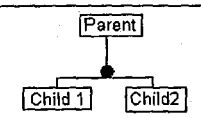
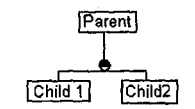
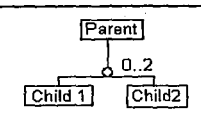
- Mandatory features: In a configuration if a feature must be selected from a feature tree whenever its parent is selected this kind of features is called conditional mandatory features. To put it simple, we call it mandatory feature thereafter.

- Full mandatory features: The root feature of a feature tree is a full mandatory feature. For a given mandatory feature in a feature tree there exists a path connecting the feature to the root. If all the nodes on the path are mandatory features the given feature is a full mandatory feature. Full mandatory features are common feature that will be included in every product in a software product line.

- Variable features: For a given feature in a feature tree when its parent is selected in a configuration whether or not this feature will be selected depends on the multiplicity and dependency constraints. This kind of features is called variable features. Variable features are usually modelled as a group associated with a multiplicity to form a variation point. The selection of variable features at a variation point will be first constrained by the selection of its parents, i.e., if and only if the parent is selected the selection of some variants from the group can then be done based on the multiplicity and dependencies between variable features.

Based on the above discussion, the number of features types in feature models can be classified to three, full mandatory features, mandatory features, and variable features. Three kinds of decomposition edges are used to distinguish the three kinds of features, full mandatory decomposition edge, mandatory decomposition edge, and variation decomposition edge. The notations used to represent these decomposition edges have been presented in Table 1. An edge attached with a solid circle is a full mandatory decomposition edge. The solid circle end of the edge connects a set of child features that are full mandatory features decomposed by their parent. An edge attached with a semi-filled circle is a mandatory decomposition edge. The semi-filled circle end of the edge connects a set of child features that are mandatory features decomposed by their parent. An edge attached with a hollow circle is a variation decomposition edge. A variation point is represented by a variation decomposition edge, a parent and a set of variable child features, called variants, connected by the edge, and a multiplicity. A multiplicity of a variation point contains two integers specifying the minimum and maximum number of children that can be selected from the variation point in product configurations. By changing the multiplicity we can represent different types of variability, such as alternative features (1..1), multiple alternative features (1..n), and optional features (0..1).

Table 1. Decomposition edges and diagrams.

Decomposition edge	Decomposition diagram
Full mandatory decomposition edge	
Mandatory decomposition edge	
Variation decomposition edge	

Based on the multiplicity associated with each variation point we define the following two kinds of variation points:

- Mandatory variation points: if the multiplicity of a variation point is in the form of 1..n (n >= 1) it is called mandatory variation points. Whenever its parent is selected in a product configuration some of the variants of this variation point must also be selected.

- Optional variation points: if the multiplicity of a variation point is in the form of 0..n (n >= 1)

it is called optional variation points. When the parent of the variation point is selected in a product configuration the variants of this variation point may or may not be selected.

3 Dependency transformations

As discussed before features and variation points have different types, such as full mandatory feature, mandatory feature, variable feature, mandatory variation point, and optional variation point. Dependencies in feature models may occur between features, or between variation points, or between a feature and a variation point. Combining all these factors together we can generate a long list of dependency types. These different types of dependencies make the product configurations and validations of the dependencies very complex and unmanageable. An approach is proposed to transform all these kinds of dependencies to a single type of dependencies, i.e. dependencies between variable features. We first study the following dependency types and then extend our approach to other dependency types later in the paper.

- dependencies between full mandatory features
- dependencies between mandatory features
- dependencies between mandatory variation points
- dependencies between optional variation point

We will discuss each of the above dependencies and show how these dependencies can be transformed to a unique type of dependencies – dependencies between variable features.

3.1 Dependencies involving full mandatory features

If a full mandatory feature requires another feature, whether it is a full mandatory, mandatory feature, or variable feature, the required feature should become full mandatory. If a full mandatory feature excludes another feature, whether it is a full mandatory, mandatory feature, or variable feature, the excluded feature should not exist in the feature model. Thus, any `Requires` and `Excludes` dependencies that include full mandatory features should not exist in feature models as `Requires` dependencies including full mandatory features in a feature model have already been supported and `Excludes` dependencies including full mandatory features are not valid and should be removed from feature models.

3.2 Dependencies between mandatory features

Dependencies between mandatory features can be transformed to the dependencies between the nearest variable ancestors of the two mandatory features. Taking Figure 1 as an example, assuming there is a dependency of "D requires J". Both D and J are mandatory features. This dependency can be transformed to a dependency of "A requires L", where A is the nearest variable ancestor of D and L is the nearest variable ancestor of J. According to the definition of mandatory feature whether or not a mandatory feature will be selected in a configuration will depend on whether or not its parent is selected. When variable feature A is selected mandatory feature D will be selected. As A requires L and A has been selected L will be selected. When L is selected mandatory feature J will be selected. Therefore, "A requires L" implies "D requires J".

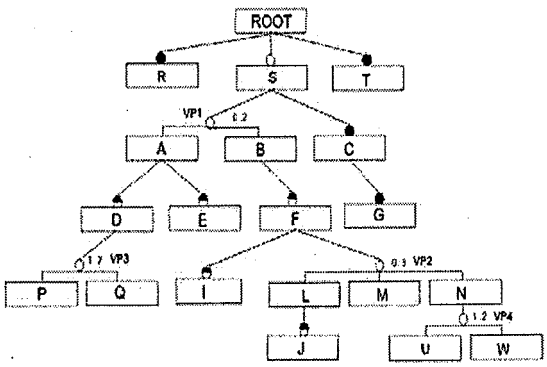


Figure 1. A feature model

3.3 Dependencies between mandatory VPs

Dependencies between mandatory variation points can be transformed to the dependencies between the parents of the variation points. For example, assume that VP3 requires (excludes) VP4 in Figure 1. This dependency can be transformed to the dependency of "D requires (excludes) N". Whenever D is selected some variants at VP3 will be selected. As D requires (excludes) N when D is selected N will be (not be) selected. When N is (is not) selected some variants at VP4 will be (not be) selected. Therefore, "D requires (excludes) N" implies "VP3 requires (excludes) VP4". Please note that "D requires (excludes) N" is a dependency between a mandatory feature and a variable feature. Based on the discussion in Section 3.2 mandatory feature D in this dependency can be replaced by its nearest variable ancestor, A. Thus the dependency "D requires (excludes) N" can be further transformed to

a dependency between variable features – "A requires (excludes) N".

Based on the above discussion any dependencies between mandatory variation points can be transformed to the dependencies between their parents. If the parents are not variable features, the dependencies can be further transformed to dependencies between the nearest variable ancestors of the parents.

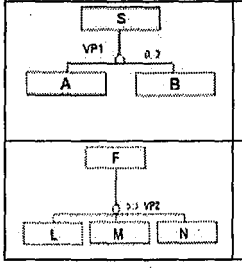
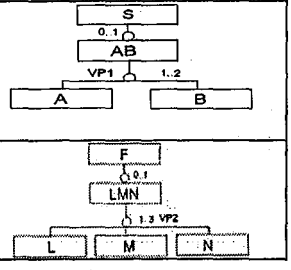
If there is a dependency between two mandatory variation points and either parent of the two variation points is a full mandatory by using the transformation method we can get a dependency between the parents of the variation points. This dependency between the two parents is a dependency involving full mandatory features. As discussed in Section 3.1 any `Requires` and `Excludes` dependencies that include full mandatory features should not exist in feature models. Based on this we can validate the dependencies between two mandatory variation points where either of the parents is a full mandatory feature, i.e. `Requires` (`Excludes`) dependencies between two such variation points are always supported (not permitted) in feature models.

3.4 Dependencies between optional VPs

Dependencies between optional variation points cannot be directly transformed to the dependencies between their parents. For example, assume that VP1 requires VP2 in Figure 1. Both VP1 and VP2 are optional variation points. This dependency means that if some of the variants at VP1 is selected some variants at VP2 must also be selected irrespective of which variants are selected. If we transform this dependency to a dependency between their parents as "S requires F". When S is selected and some variants at VP1 are also selected the set of variants at VP2 may or may not be selected even if its parent F is selected. Thus, "S requires F" does not imply "VP1 requires VP2".

To solve this problem, we introduce a level of abstraction between the parent and the variants for optional variation points. A new variable parent feature will be introduced to the set of variants at each optional variation point. For example, for the optional variation points VP1 and VP2 in Figure 1, this kind of changes has been shown in Table 2. The changed variation points have the same configuration implications as before. Taking VP1 as an example, at the original VP1, based on the multiplicity of 0..2, when S is selected there are four possible choices at this variation point, selecting feature A, selecting feature B, selecting both feature A and B, selecting nothing. In the changed VP1, a new optional variable feature AB is introduced with a multiplicity of 0..1 and the multiplicity of the original VP1 has been changed from 0..2 to 1..2. When S is selected feature AB may or may not be selected. When AB is selected, based on the multiplicity of 1..2 we can select either Feature A, or Feature B, or both A and B. When AB is not selected, both A and B will not be selected. As the configuration implications of the original VP1 and Changed VP1 are the same they can be considered equivalent with each other.

Table 2. changed variation point representation

Original VP	Changed VP
	

Based on the abovementioned method we have changed the feature model shown in Figure 1 to the model shown in Figure 2. The original dependency of "VP1 requires VP2" can then be transformed to a dependency between the newly added variable parent features of "AB requires LMN". When variable feature AB is selected some of the variants at VP1 will be selected. As AB requires LMN and AB has been selected LMN will be selected. When LMN is selected some of the variants at VP2 will be selected. Therefore, the dependency of "AB requires LMN" is equivalent to the dependency of "VP1 requires VP2". Based on the example, any dependencies between two optional variation points can be transformed to dependencies between variable features by introducing a level of abstraction.

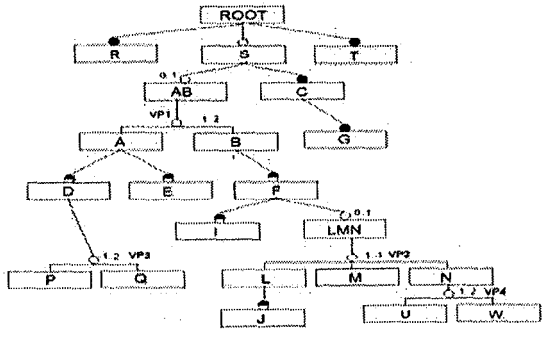


Figure 2. Changed feature model based on Figure 1.

Based on the discussions in Sections 3.2 – 3.4 we can summarise the transformation rules for different types of features and variation points as follows.

- A mandatory feature involved in a dependency can be replaced by its nearest variable ancestor
- A mandatory variation point involved in a dependency can be replaced by its parent. If the parent is not a variable feature it can be further replaced to its nearest variable ancestor.
- An optional variation point involved in a dependency can be replaced by its newly added optional variable parent as discussed in Section 3.4. The multiplicity of the variation point will be changed from 0..n to 1..n.

Based on the above rules we can easily transform the following dependency types to dependencies between variable features.

- dependencies between a mandatory feature and a variable feature
- dependencies between a mandatory feature and a mandatory variation point
- dependencies between a mandatory feature and an optional variation point
- dependencies between a variable feature and a mandatory variation point
- dependencies between a variable feature and an optional variation point
- dependencies between a mandatory variation point and an optional variation point.

For example, assume that there is a dependency "D requires VP2" in Figure 1. This is a dependency between a mandatory feature and an optional variation point. This dependency means that if D is selected some variants must also be selected at VP2, irrespective of which variants are selected. According the rules presented above mandatory feature D in this dependency can be replaced by its nearest variable ancestor, A, and VP2 can be replaced by its variable parent that is a newly added optional variable feature, LMN. Then the dependency "D requires VP2" can be transformed to the dependency "A requires LMN" in Figure 2. When A is selected D will be selected. As D requires LMN, LMN will be selected. When LMN is selected some of the variants at VP2 will be selected irrespective of which variants are selected. Therefore, the dependency of "A requires LMN" implies the dependency of "D requires VP2".

4 Case study

We have applied the approach to a Library Software Product Line. Because of limited space of this paper we cannot show the whole feature model. A much simplified feature model with some features omitted is shown in Figure 3. A complete feature model can be found in [7]. Different kinds of dependencies in the model have been listed in Table 3. We then apply the approach to transform all these dependencies in dependencies between variable features. The dependencies after the transformation are also listed in Table 3. The feature model after the transformation is shown in Figure 4.

The configuration implications of the dependencies before and after transformation are the same. But the feature model after the transformation has the following advantages.

- The new added variable feature as the parent for an optional variation point adds a level of abstraction to the group of variants of the variation point, which makes the semantics of the group of variants more intuitive and comprehensive. For example, in Figure 4, the new parent of VP7, Network security, specifies the high level functionality of its variants, Message encryption and Firewall.
- The semantics of the transformed dependencies are much more understandable. For example, the original dependency VP2 requires VP6 is transformed to Network based (system) requires On-line payment. It is obvious that the semantics of the new dependency makes much more sense for software engineer to configure products.
- The configuration constraint of each dependency is simplified, which makes the validation of the dependencies in a feature model much easier. For example, configuration constraint of the original dependency VP2 requires VP6 can be represented as "**If** Lan based ∨ Internet based **then** Bill pay ∨ PayPal ∨ (Bill pay ∧ PayPal)" . But this constraint can be simplified by the transformed dependency "Network based requires On-line payment" as "**If** Network based **then** On-line payment".

As the configuration constraint of each dependency is simplified the validations of all the dependencies in a feature model become easier. How to validate dependencies in feature models is out of the scope of this paper.

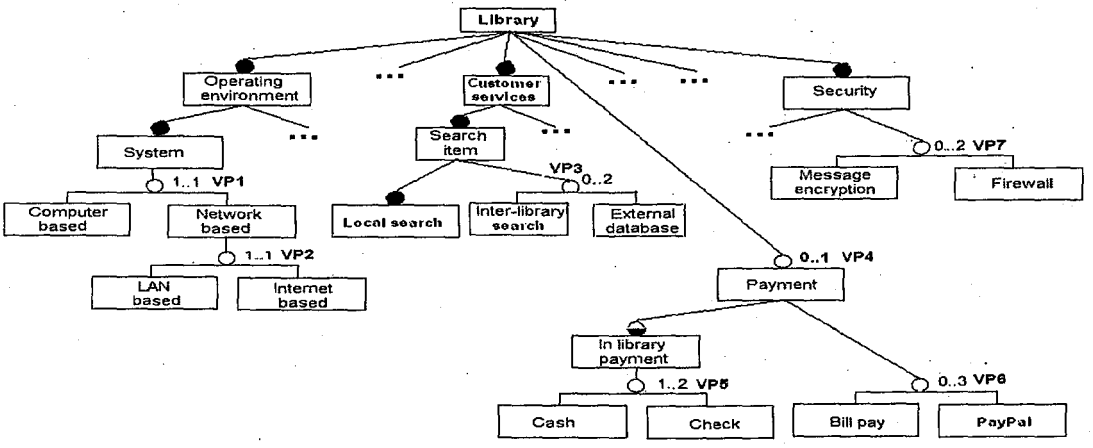


Figure 3. A feature model for the Library Software Product Line (before feature dependency transformation).

Table 3. Dependencies before and after transformation.

Dependencies before transformation		Dependencies after transformation
Dependency type	Dependency	
Variant-Optional VP	Computer based excludes VP7	Computer based excludes Network security
Variant-Optional VP	Network based requires VP7	Network based requires Network security
Variant- Mandatory VP	Computer based requires VP5	Computer based requires payment
Mandatory VP – Optional VP	VP2 requires VP6	Network based requires On-line payment
Optional VP - Variant	VP6 requires Message encryption	On-line payment requires Message encryption
Mandatory VP – Optional VP	VP2 requires VP3	Network based requires Remote search
Variant – Optional VP	Computer based excludes VP3	Computer based excludes Remote search
Variant – Optional VP	Computer based excludes VP6	Computer based excludes On-line payment

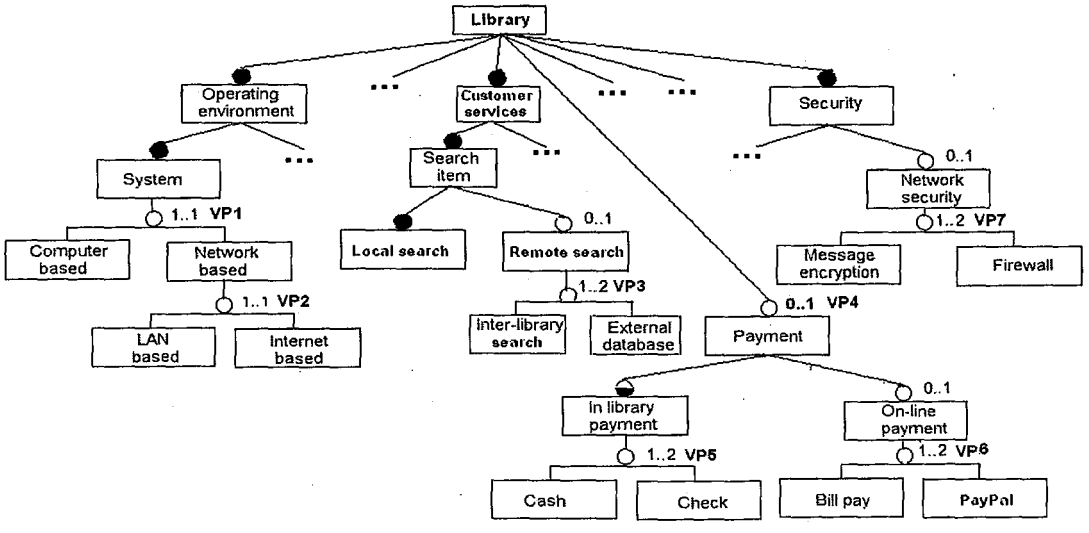


Figure 4. The Library Software Product Line feature model after feature dependency transformation.

5 Conclusion

Feature dependencies have great implications on product configurations in software product lines. Current existing dependency types make the dependency validation and product configuration very complex. An approach to streamlining dependency types in software product lines is proposed. Different types of dependencies can be transformed to a single type of dependencies between variable features by employing the approach. A case study based on the Library Software Product Line has been conducted to demonstrate how the approach works. As the constraint of each transformed dependency has been simplified the validation of the dependencies becomes much simpler. The semantics of the dependencies after transformation are more intuitive and comprehensive, which would make the product configurations more efficient and effective.

6 Acknowledgement

This work is supported by Australian Research Council Discovery Project (DP 0772799).

7 References

[1] Kang, K., Cohen, J., Novak, W., and Peterson, S. "Feature-Oriented Domain Analysis Feasibility Study". Technical Report CMU/SEI-90-TR-21, Software Engineering Institute, Carnegie Mellon University, November 1990.

[2] Griss, M., Favaro, J., and dlAlessandro, M. "Integrating Feature Modeling with the RSEB". In *Proceedings of the Fifth International Conference on Software Reuse,* pages 76-85, Vancouver, BC, Canada, June 1998.

[3] Czarnecki, K. and Eisenecker, U.W. "Generative Programming: Methods, Tools, and Applications". Addison-Wesley, Boston, MA, 2000.

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

[5] Trinidad, P., Benavides, D., Duran, A., Ruiz-Cortes, A., and Toro, M. "Automated error analysis for the agilisation of feature modelling". *Journal of Systems and Software*, Volume 81, pp. 883–896, 2008.

[6] Schobbens, P., Heymans, P., and Trigaux, J. "Feature Diagrams: A Survey and a Formal Semantics". In *Proc. of 14th IEEE International Requirements Engineering Conference*, 2006.

[7] Lin, Y., Ye., H., and Tang, J. "An Approach to Efficient Product Configuration in Software Product Lines". Accepted by the 14th International Software Product Line Conference, SPLC 2010.