Exploring the Commonality in Feature Modeling Notations

Miloslav ŠÍPKA*

Slovak University of Technology
Faculty of Informatics and Information Technologies
Ilkovičova 3, 842 16 Bratislava, Slovakia
miloslav.sipka@gmail.com

Abstract. Feature modeling is an important approach to capturing commonalities and variabilities in system families and product lines. This article analyzes the commonalities and variabilities among different approaches to feature modeling with respect to their use in order to determine how feasible is to unite their notations into a consistent unit and support them in a single feature modeling CASE tool. Suggested notation basis combines the extended Czarnecki-Eisenecker notation (with UML like cardinalities) with Gurp-Bosh-Svahnberg edge decorations and identifies mandatory parts of the information associated with features.

1 Introduction

Feature modeling is the activity of modeling the common and the variable properties of concepts and their interdependencies and organizing them into a coherent model referred to as a feature model [1].

A *feature model* represents the common and the variable features of concept instances and the dependencies between the variable features. Model represents the intention of a *concept*, whereas the set of instances it describes is referred to as the extension of the concept. A feature model consists of a feature diagrams and some additional information such as short semantic description of each feature, rationale for each feature, constraints, default dependency rules etc.

A *feature diagram* consists of a set of nodes, a set of directed edges, and a set of edge decorations. The nodes and the edges form a tree. he edge decorations are drawn as arcs connecting subsets or all of edges originating from the same node.

---

* Supervisor: Ing. Valentino Vranić, PhD., Institute of Informatics and Software Engineering, Faculty of Informatics and Information Technologies STU in Bratislava

M. Bieliková (Ed.), IIT.SRC 2005, April 27, 2005, pp. 139-144.

A *mandatory feature* is included in the description of a concept instance if and only if its parent is included in the description of the concept. If the parent of a mandatory feature is optional, the mandatory feature should not be part of the description. A mandatory feature node is pointed to by a simple edge optionally ending with a filled circle (**Fig. 1** row 1).

An *optional feature* may be included in the description of a concept instance if the parent is included. If the parent is not included, the optional feature cannot be included. An optional feature node is pointed to by a simple edge ending with an empty circle (**Fig. 1** row 1).

A concept (and similarly a feature) may have one or more sets of direct *alternative features*. If the parent of a set of alternative features is included, then exactly one feature from this set of alternative features is included in the description, otherwise none. The nodes of a set of alternative features are pointed to by edges connected by an arc (**Fig. 1** row 2).

A concept (and similarly a feature) may have one or more sets of direct *or-features*. If the parent of a set of or-features is included in the description of a concept instance, then any non-empty subset from the set of or-features is included in the description, otherwise none. The nodes of a set of or-features are pointed to by edges connected by a filled arc (**Fig. 1** row 3).

Feature modeling helps us to avoid two serious problems: First, relevant features and variation points are not included in the reusable software. Second, many features and variation points are included but never used and thus cause unnecessary complexity, development cost, and maintenance cost. Finally, the feature models provide us with an abstract (since implementation independent), concise, and explicit representation of the variability present in the software.

2 A Survey of Feature Modeling Notations

Today most accepted is Czarnecki-Eisenecker notation. Its main advantages are that this notation is compatible with the original FODA notation while at the same time bringing some new possibilities. In Fig. 1 the original FODA notation and both Czarnecki-Eisenecker notations-the base and the extended one-are compared.

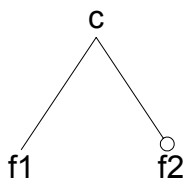
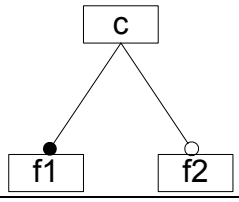
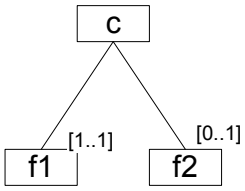
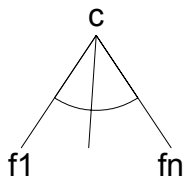
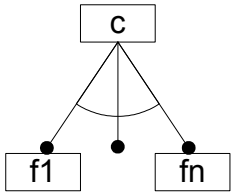
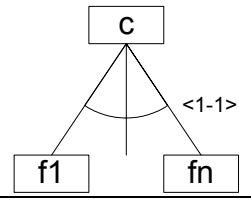
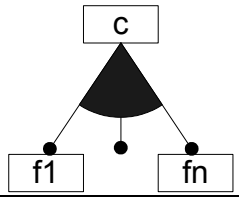
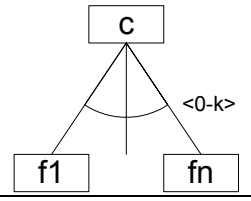
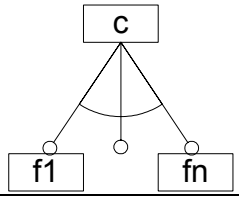
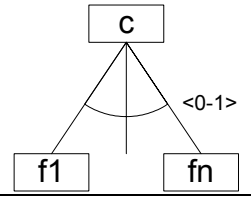
Original FODA notation [4]	**Czarnecki-Eisenecker notation**	**Extended Czarnecki-Eisenecker [2]**
mandatory and optional subfeature 	mandatory and optional subfeature 	mandatory and optional subfeature 
alternative subfeatures 	XOR group 	group with cardinality <1-1> 
	OR group 	group with cardinality <0-k> 
	XOR group with optional subfeatures 	group with cardinality <0-k> 

Fig. 1. Comparison of feature modeling notations [2]

Another UML-based notation is the one used in FeatuRESB [3]. It explicitly deals with variation points. It defines static and dynamic binding in a variation point. This notation was overcome by Gurp-Bosch-Svahnberg notation [6] which has been developed for expressing variability in software product lines. It brings OR and XOR specialization into variation points. Initially, the binding type was not represented as edge decorations, but this can be done using the flag at the feature. In Fig. 3, the two mentioned UML-based notations are confronted with Czarnecki-Eisenecker notation.

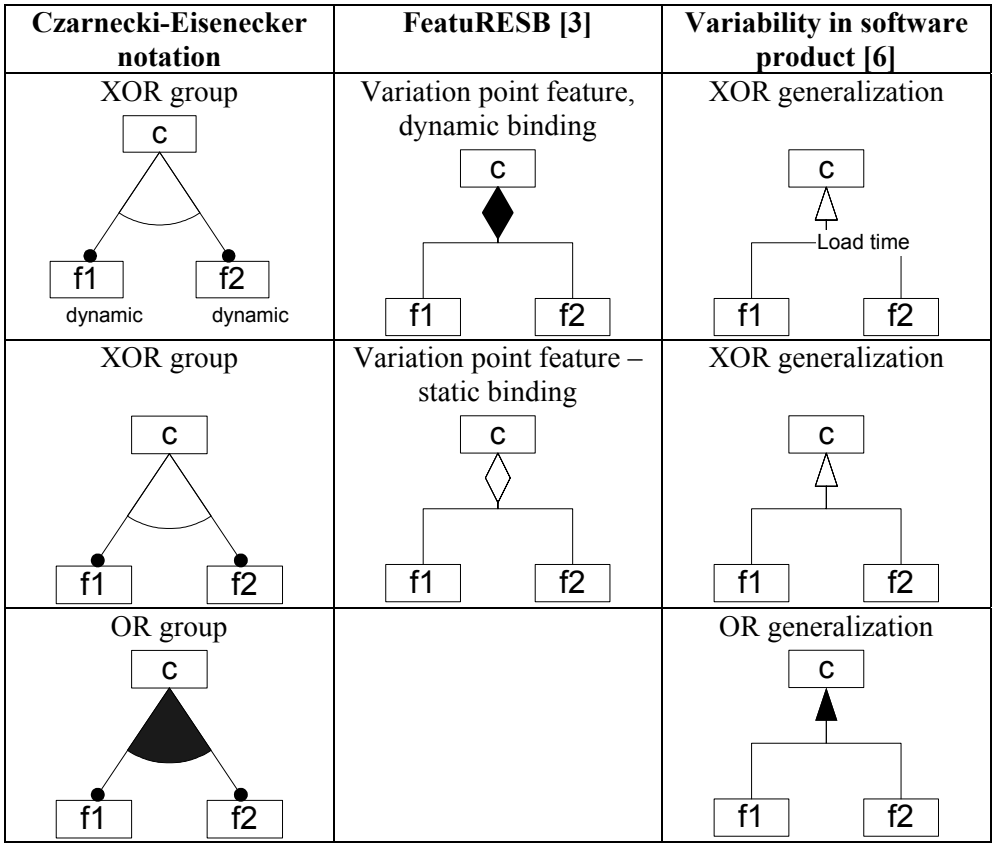


Fig. 2. Comparison of others feature modeling notations

2.1 Additional Information provided with features

First additional information provided with features is binding time and binding mode. Binding time is important if we have to describe implementation details of feature and code with references feature. So it is important to identify binding time of feature.

First of five binding time is source time. The decision to bind feature is made by programmer or designer during the implementation phase. Other binding times are compile and link time. In this case the feature is bound automatically by preprocessor or special linker program. There is no reason to make difference between linking and compile time. Some languages do not have linking at all. This three times represents static binding.

Other two types of bounding represent dynamic bounding. It is load and rum time. Feature with load time bounding is selected in instance of concept in time when the program loads in to memory. Feature with run time bounding can be selected widely when the program is running. In both cases we must implement feature and compile it.

But in fact implementing other modules can be different if feature can change in run time of the program.

Another additional information supported with feature is layer. FODA [4] identifies four layers of features. There are Operating environment layer, Capability layer, Domain technology and Implementation technique. But there can be need of more than four layers of features. So categorization of features in layers is not so important and is used very rarely.

One important categorization is to identify if feature represents abstract (identical as a abstract class in OOP) thing or have to be implemented and integrated as a part of code.

Other additional information provided with features and feature model in general are constraints and default dependency rules. Constraints represent necessary conditions each instance of concept have to fulfill. Default dependency rules represent rules of choosing default feature while creating instance of a concept. In FODA the constraints are reduced to simple rules representing the so-called requires and mutual exclusion relationships between features. A more sophisticated way of representing constrains is to use predicate logic [5].

2.2 Common parts of feature diagram notation

After comparing several notations we can extract the common parts. Czarnecki-Eisenecker notation can act as a backbone for this. The reason is simple: each of the analyzed notations extends this notation. Original Czarnecki-Eisenecker notation will be improved in two ways. The first is extending by cardinalities as it is in **Fig. 1**. The second way is to identify relationship between a feature and its subfeature as in **Fig. 2**.

2.3 Common parts of associated information

It seems there are lot of information associated with features and feature diagram. If fact it is impossible to choose set of information for common notation. The best way is let user to choose witch associated information he need. In this case it is important to let user choose what associated information will be displayed in a feature diagram and how. Truly common parts of associated information are constrains and default dependency rules. One effective way to express constraints is predicate logic [5].

3 Conclusions

In our survey we compare six notations of feature model. We discover that there are no collisions between these notations. Concerning feature diagram, each notation introduces different quantum of information in diagram. At one side, information contained in diagram doesn't have to be included as additional information. At other side feature diagram can be difficult to read. Concerning additional information, we found that each notation provides other type of information.

Further work includes developing of a metamodel that contains all described notation elements. After that implementing a support tool which will use this metamodel. So it allows user to choose how to express relations between features, either in diagram or as associated information.

Acknowledgement: This work has been partially supported by the Grant Agency of Slovak Republic grant No.VG1/0162/03.

References

1. Czarnecki, K. Eisenecker, U.W.: *Generative Programming: Principles, Techniques and Tools*. Addison-Wesley, 2000.
2. Czarnecki, K. Helsen, S. Eisenecker, U.W.: Staged Configuration Using Feature Models. In: *SPLC 2004, LNCS 3154*, R.L. Nord (Ed.), 2004, 266–283.
3. Giss, M. L. Favaro, J. d'Alessandro, M.: Integrating Feature Modeling with the RSEB. In: *Proc. of 5th International Conference on Software Reuse*, Vicoria, B.C., Canada. IEEE Computer Society Press (1998), http://www.favaro.net/john/home/publications/rseb.pdf
4. Kang, K.C., Cohen, S.G., Hess, J.A., Novak, W.E., Peterson, A.S.: *Feature-oriented domain analysis (FODA): A feasibility study*. Technical Report CMU/SEI-90-TR-21, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, USA, November 1990.
5. Vranić, V.: *Multi-Paradigm Design with Feature Modeling*. PhD thesis, Slovak University of Technology in Bratislava, 2004.
6. Group, J., Bosh, J., Svahnberg, M.: On the Notion of Variability in Software Product Lines. In: *Proceedings of WICSA 2001* (2001), http://www.jillesvangurp.com/publications/notionOfVariability.pdf.