

Feature interaction and composition problems in software product lines

Silva Robak

Technical University Zielona Góra, Institute of Computer Science and Management
robak@pz.zgora.pl

Bogdan Franczyk

Intershop Software Entwicklungs GmbH, Intershop Research
b.franczyk@intershop.de

Abstract

Features are essential characteristic of applications within a product line. Features organized in different kinds of diagrams containing hierarchies of feature trees are closely related to variation points, which appear at different levels and life cycle phases for product lines. Optional and alternative variants attached to variation points may be additionally constrained by mutual-exclusion or –inclusion and classified by binding modes, times and sites. Variability leads further to crosscutting variability. Reducing an n:m-relation between feature and variation point would help organizing scalable and traceable software models according to SoC with less intertwined feature trees.

Keywords : feature composition, feature interaction, crosscutting features, product lines

Workshop Goals: identify and categorize characterizing properties of various instances of feature interaction problems

Working Groups: feature-based methods for product lines, integrating UML with feature-based methods, handling variability at different levels

Background

Building the sets of related systems helps to achieve the remarkable gains in productivity and to improve time-to-market and product quality [Clements 99]. A development of a group of systems made from common generic software assets is to obtain by building a software product lines upon software product families [Czarnecki 00]. Features are essential characteristic of applications within a product line. Domain Analysis methods [Arrango 94] like feature analysis are on a high level of abstraction and provide a concise and explicit representation of commonality and variability [Coplien 99] contained in product family.

Features may be organized in different kinds of feature diagrams (essential for reuser), containing hierarchies of feature trees (graphs) with mandatory, optional and alternative features [Kang 90]. A root of the tree represents a concept being described and the remaining nodes denote features. Mandatory features have to be always included in every system instance, an optional feature may be included or not, and an alternative feature replaces another feature when included. Mandatory features which parent-features are neither optional nor connected to alternatives represent the common features that each family member possesses. The rest of the features are known as variant features, which represent the permissible differences (deltas) between family members. A part of the implementation process of a member is a selection of these variants. The means like configuration tables

containing the variable features of the member additionally describe the choices, which can be made.

Optional and alternative features are closely related to variation points [Jacobson 97] and may appear at different levels and phases in software development process Domain Engineering (DE) or Application Engineering (AE) for product lines. The later the variability points (seen as “delayed decisions”) would be introduced to a system, the more different systems could be build [Gurp2000].

Variants attached to variation points lead to a set of combinations, but only a subset of it may be correct and leads to complete configurations. The problems associated with feature specification and interaction (i.e. when one feature modifies or influences another features) have been discussed since early nineties [Zave 93]. Variability may be resolved with different techniques, especially efficient with those according to Separation of Concerns (SoC) principle.

Position

Our position is that:

1. There is no concise feature definition, because features of various granularities are requirements (functional and non-functional) and properties of product lines as software intensive systems. The features may be described in different models and views and depend on stakeholders attached to the view (e.g. customer, system analysts and designer, user, end user). The products in a product line are sharing a common, managed set of features that satisfy specific needs of selected market or mission, while product families share features of more technical nature.
2. Essential features may be organized in different kinds of diagrams containing hierarchies of feature trees. Commonly used kinds of feature trees [Kang 90, Kang 98, Griss 98, Czarnecki 00, Gurp 00] are alone not able to express various possible feature’s roles dependencies and relationships. Complementary notations for enhancing semantic of the domain models (like described in [Hein 00]) are needed.
3. Decisions about what will be common and what the variable parts in a software product line have more strategic than technical nature and can therefore change over time. Commonality expressed as mandatory features constraint the size of the software family. Variability as optional or alternative features enlarges the family size, but as generic (i.e. parameterized) places also increase systems complexity. Good design (e.g. according to SoC) helps to avoid crosscutting variability when product line evolves.
4. Optional and alternative feature are closely related to variation points, which may appear at different levels and life cycle phases for product lines. The variability has to be resolved within a particular part of the DE or AE and by decision-makers responsible for it.
5. In feature diagrams optional and alternative variants attached to variation points may be additional constrained by or mutual exclusion (strong form of conflicting [Bosch 00]) or mutual inclusion. Dependencies and composition rule for features may be partly described within feature trees with UML-like relationships (i.e. composed_of,

generalization/ specialization, implemented_by) [Kang98], [Bosch 00], and other means like tables and matrixes, textual descriptions as constraints.

6. Features may be further classified by binding modes, times and sites. Binding time may be classified as construction- time, installation- and use-time. Construction- time (also referred to as a build time) includes the source time (pre-compile time), compile, link and load time. For generative techniques a generation time may be seen as an additional kind of binding time within a build time.
7. Variability at certain design stage leads further to crosscutting variability in later stages. Reducing an n:m-relation between feature and variation point would help organizing scalable and traceable software models according to SoC with less intertwined feature trees. The usage of the generative techniques like Aspect-Oriented Programming AOP [Kiczales 97], Subject-Oriented Programming SOP [SOP] or frame engineering [Basset 97] may play the special role there.
8. Object-oriented technology notations like UML diagrams should be used together with feature models for modeling variants during analysis.

Approach

We can treat features as abstractions of the requirements, too. A particular requirement may apply to several features and a particular feature can be required to fulfill more than one requirement (a n:n-relation) [Gurp2000]. There is usually a 1:n-relation between a feature and its implementation. A feature implementation may be usually spread through many assets (“crosscutting features”), except using special techniques like AOP or SOP “separating concerns”.

Domain analysis methods are used to identify and grouping sets of features. The FORM approach [Kang 98] is an extension to FODA [Kang 90] and provides methodology for developing domain architectures and components for reuse upon feature models. Like FeatuRSEB [Griss 98] extended the UML-based RSEB method [Jacobson 97] with the feature model, the integration of feature oriented methods with the object-oriented technology seems to be the next step towards supporting product line development.

The shortcoming of the feature trees is the restriction of variability in feature specification to some binding times and a decomposition type. There can be also other attributes needed to choose a variant like availability sites (i.e. when, where and to whom a feature can be available), variability mechanisms, and binding modes (e.g. static, dynamic), binding occurrence, descriptions, etc.

There are many different approaches for solving variability at the code level. The chosen technique should possible support the SoC-principle and provide the maintainability (traceability) and the scalability of applications. Features should be forwards and backward traceable with different tools helping untangling feature maps from features to products [Griss 00].

Handling variability at the code level

There are several approaches to handling product-line variability at a various levels of abstraction, according to different binding times and associated with them a generic assets

representation. The common object-oriented techniques are abstractions, the different kinds of inheritance, the overloading, and the aggregation with the delegation and also the parameterization. Some of the techniques were derived from a particular programming language and are not available everywhere (i.e. overloading and multiple inheritance is not available in Smalltalk or in Java). Another new techniques like AOP or Dynamic Class Libraries are not obtainable for older programming language (i.e. for C++). The object-oriented techniques like aggregation with delegation are used in design patterns. Application frameworks using design patterns may be employed in product line context since they provide solutions for managing the variations. Mechanisms like static libraries, conditional compilation and generative techniques like frame technology [Basset 97] are possible to use for all programming languages.

The crosscutting variability that affects many components may be difficult to handle.

References

[Arrango 94] Arrango G., *Domain Analysis Methods*. In Software Reusability, Schäfer, Prieto-Díaz R., and Matsumoto M. (Eds.), Ellis Horwood, New York, New York, 1994, pp. 17-49.

[Basset 97] Basset P., *Framing Software Reuse - Lessons from Real World*, Yourdon Press, Prentice Hall, 1997.

[Bosch 00] Bosch J., *Design and Use of Software Architectures. Adopting and evolving product-line approach*. Addison-Wesley, 2000.

[Clements 99] Clements P., Northrop L.M., *A Framework for Software Product Line Practice - Version 2.0* [online]. Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, July 1999. <URL: <http://www.sei.cmu.edu/plp/framework.html>>

[Coplien 99] Coplien J., *Multi-Paradigm Design for C++*, Addison-Wesley, 1999.

[Czarnecki 00] Czarnecki K., Eisenecker U., *Generative Programming Methods, Tools and Applications*, Addison-Wesley, 2000.

[Griss 98] Griss M. L., Favaro J., D'Alessandro M., *Integrating Feature Modelling with the RSEB*. Proceedings of ICSR98, Victoria, BC, IEEE, June 1998, pp. 36-44.

[Griss 00] Griss M. L., *Implementing Product-Line Features with Component Reuse*. 6th International Conference, proceedings/ICSR-6, Vienna, Austria, June 27-29, 2000; In William B. Frakes (ed.) "Software Reuse: Advances in Software Reusability". Springer, pp. 137-152.

[Gurp 00] Gurp J., Bosch J., Svahnberg M., *Managing Variability in Software Product Lines*. Landelijk Architectuur Congres 2000.

[Hein 00] Hein A., Schlick M., Vinga-Martins R., *Applying Feature Models in Industrial Settings*. Proceedings of the The First Software Product Line Conference (SPLC1). Denver, Colorado, USA, pp.47-70, 2000.

[Jacobson 97] Jacobson I., Griss M. L. and Jonsson P., *Software Reuse Architecture, Process and Organization for Business Success*, Addison-Wesley, 1997.

[Kang 90] Kang K., Cohen S., Hess J., Nowak W., and Peterson S., *Feature-Oriented Domain Analysis (FODA) Feasibility Study*. Technical Report No. CMU/SEI-90-TR-21, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania 1990.

[Kang 98] Kim S.J., Lee J., Kim K.J., Shin S.H., and Huh M.H., *FORM: A Feature-Oriented reuse Method with Domain specific reference architectures*. Annals of Software Engineering, Vol. 5, pp.143-168, 1998.

[Kiczales 97] Kiczales, G., Lamping, J., Mendhekar, A., Maeda, C., Lopes, C. V., Loingtier, J. M., Irvin, J., *Aspect-Oriented Programming*, Proceedings ECOOP97 – 11th European Conference of Object-Oriented Programming, Jyväskylä, Finland, June 1997, Mehmet Aksit and Satoshi Matsuoka (Eds), LNCS 1241, Springer-Verlag, 1997.

[SOP] Homepage of the Subject-oriented Research Project, IBM, Thomas J. Watson Research Center, Yorktown Heights, NY, <URL: <http://www.research.ibm.com/sop>>

[Zave 93] Zave P., *Feature interactions and formal specifications in telecommunications*, IEEE Computer XXVI (8): pp.20-30, August, 1993.