By Jaejoon Lee and Dirk Muthig

FEATURE-ORIENTED VARIABILITY MANAGEMENT IN PRODUCT LINE ENGINEERING

Implementing feature-oriented variability modeling throughout the life cycle.

Most organizations selling software-intensive products today face a similar challenge: providing high-quality products of increasing size and complexity customized to the special needs of individual customers or market segments with less amounts of effort and time. Mastering this challenge requires mature and adequate engineering and management practices focusing on future needs of customers or markets. Product line approaches, such as Fraunhofer PuLSE (Product Line Software and System Engineering) provide the required methods, processes, and tools for planning, engineering, managing, and maintaining products of an organization as a family characterized by well-defined sets of common and varying properties. Efficiency is then achieved through systematic reuse that has been proactively planned with respect to expected future requirements.

Expected products for targeting market segments are captured explicitly by a product line scope that is intensively discussed among all organizational units including marketing, management, and development. The scope thus captures a common view of an organization's future and thus specifies the type of products, as well as the space of features and their variations, that should be supported by a reuse infrastructure.

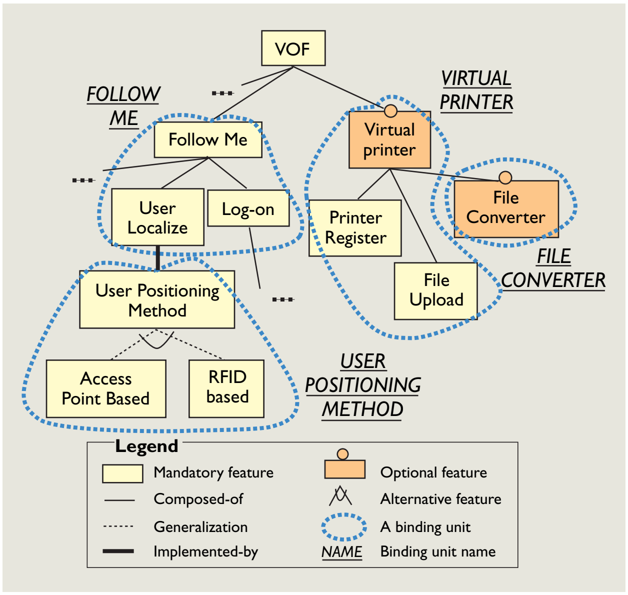


Figure 1. A feature model of a VOF product line.

From our experience in applying Fraunhofer PuLSE with diverse industry partners since 1998, variability management, which consistently covers all life cycle stages and organizational roles, is fundamental to run a sustainable successful product line in practice. Variability management is the major product-line-unique discipline that must be newly established within non-product-line organizations. It is responsible for systematically managing the scope itself, and ensuring its traceability with genericity of product line artifacts.

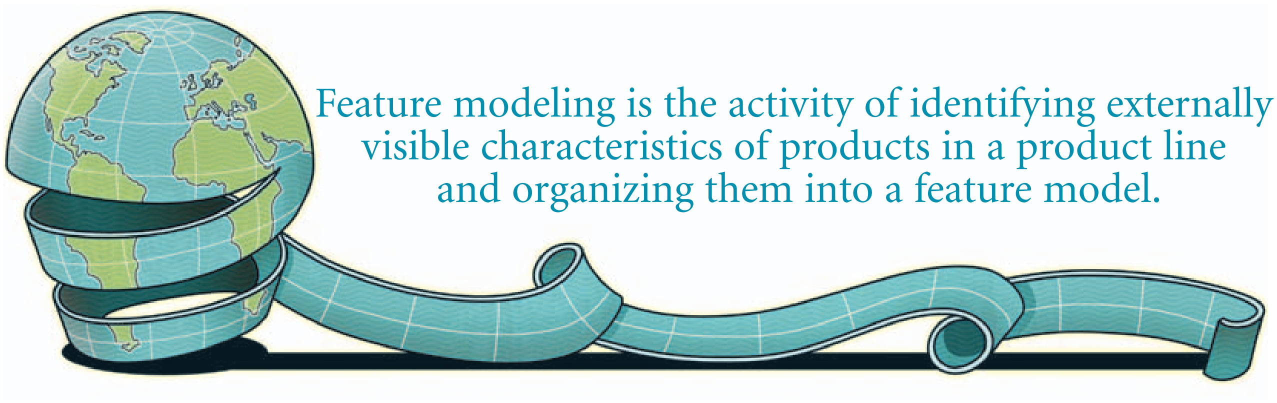
This article introduces a feature-oriented approach to explicit modeling and managing variability information. It takes features as first-class entities for controlling variability and enables an organization to exploit commonality and manage variability in both problem and solution space.

Feature Orientation

The idea of feature orientation for analyzing commonality and variability of a product line appeals to many product line engineers because features are effective "media" supporting communication among diverse stakeholders of a product line. Therefore, it is natural and intuitive for people to express commonality and variability of product lines in terms of features.

Especially envisioned, future products are typically discussed and described in terms of features gathered from market surveys, individual customers, research labs, or technology roadmaps. For constructing a successful product line, however, more detailed information on features is required. Therefore, a feature model is refined to provide a more tangible basis for a later development, parameterization, and configuration of various reusable assets (for example, product line requirement models, reference architectural models, and reusable code components).

Feature orientation, as such, has been used extensively for product line engineering both in industry and academia, after the Software Engineering Institute first introduced Feature-Oriented Domain Analysis (FODA) as early as in 1990 [6]. For example, Organization Domain Modeling (ODM) [11] builds on both the faceted classification approach and FODA, and FeatuRSEB [5] incorporated the FODA's feature model as a catalog of commonality and variability captured in models (such as use case and object models). Feature-Oriented Reuse Method (FORM) [7] also extended the original FODA to address the issues of reference architecture development and object-oriented component engineering. Moreover, a feature model is used in con-



Feature modeling is the activity of identifying externally visible characteristics of products in a product line and organizing them into a feature model.

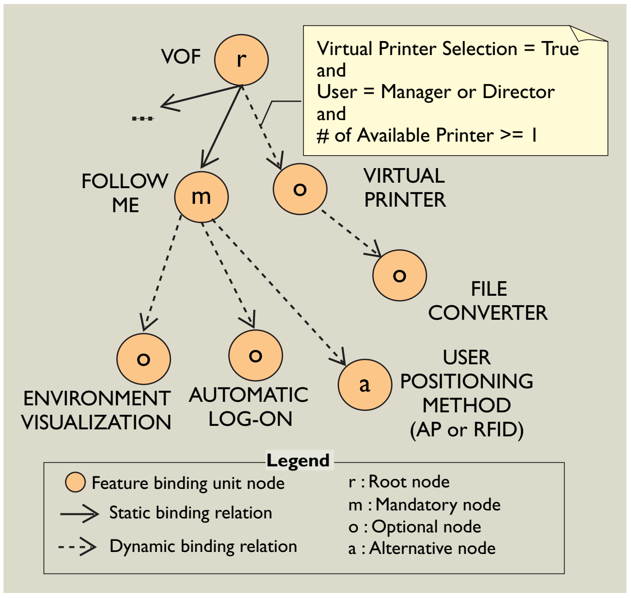


Figure 2. A feature binding graph.

junction with aspect-oriented programming [8], generative programming [2], formal methods [1], and reengineering of legacy systems [3].

Here, we introduce some basic concepts of feature modeling and extensions that we made to address new challenges originated from an ambient applications product line.

Feature Modeling: Basic Concepts and Extensions

Feature modeling is the activity of identifying externally visible characteristics of products in a product line and organizing them into a feature model. Features can be services (such as call forwarding in the telephony product line), operations (dialing in the telephony product line), nonfunctional characteristics (performance), and technologies (navigation methods in the avionics product line) of a particular product line.

Common features among different products are modeled as mandatory features, while different features among them may be optional or alternative. Optional features represent selectable features for products of a given product line and alternative features indicate that no more than one feature can be selected for a product. A feature diagram, a graphical AND/OR hierarchy of features, organizes identified features by using three types of relationships: composed-of, generalization/specialization, and implemented-by. Composition rules supplement the feature model with mutual dependency and mutual exclusion relationships that are used to constrain the selection from optional or alternative features.

Figure 1 shows a feature model of a Virtual Office of the Future (VOF) product line, which provides a range of various office services (including printing, organizing a meeting, planning a business trip, and so forth) to a user virtually anywhere in an office building whenever the user requests; see www.ricoh.rlp-labs.de/index.html. The VOF product line includes service features (Virtual Printer, Follow Me, Smart Fax), operational features (Logon, User Localize), and technology features (User Positioning Method).

A feature model is then analyzed to capture feature binding units (FBUs) [9]; an FBU is defined as a set of features that are related to each other by the relationships in a feature model (see Figure 1 for the identified FBUs). Features that belong to the same FBU work for a common service and, therefore, must exist together for correct operation of the service. This grouping reduces complexity in managing variations of product configuration, thus helping engineers to analyze the change impacts of a feature selection. Also, binding dependency between variation points can be identified and managed efficiently with mappings to FBUs. For instance, it became clear that FILE CONVERTER could be selected only after its parent FBU, VIRTUAL PRINTER, was selected.

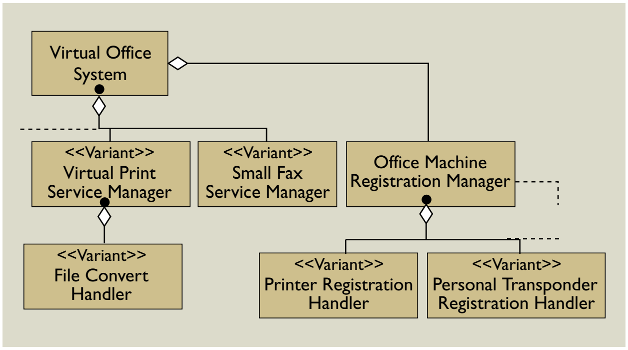

Virtual Office System
<<Variant>> Virtual Print Service Manager
<<Variant>> Small Fax Service Manager
Office Machine Registration Manager
<<Variant>> File Convert Handler
<<Variant>> Printer Registration Handler
<<Variant>> Personal Transponder Registration Handler

Figure 3. Excerpt of a generic design model.

Recently, there have been increasing demands for the postponement of decisions on product variations to runtime to provide an operating-context relevant service [4]. This means features can be selected and/or parameterized at runtime by a user or by a product itself when a certain contextual change is recognized. For example, the service features of VOF should be dynamically reconfigured to provide context-relevant services as users' physical locations and available resources change dynamically. To support such demands, a feature model is further enhanced with a feature binding graph, which is a labeled digraph without cycles and captures FBUs and their binding relations with preconditions. (See Figure 2 for a feature binding graph of VOF.)

In a feature binding graph, each node corresponds to an FBU identified through the feature binding analysis and the relation between two FBUs is either static or dynamic. Also, a dynamic binding relation is annotated with preconditions that indicate when two FBUs can be bound at runtime. For instance, the Virtual Printer feature, which selects the nearest printer to a user with a most appropriate printing quality at the moment when the service is requested, must be bound at runtime only if: the feature is selected for the current product configuration; the requesting user's job function is a manager or a director; and more than one printer is available nearby. (See the annotation at the dynamic binding relation between VOF and VIRTUAL PRINTER nodes in Figure 2.) With the feature binding graph, we could provide an intuitive and visual description of dynamically changing product configuration. Also, we could explore appropriate binding techniques based on this information.

FBU	Allocated Components	Location of Variation Points	Child FBU
VIRTUAL PRINTER	Virtual Print Service Manager	Virtual Office System	FILE CONVERTER
	Printer Registration Handler	Office Machine Registration Manager	

Part of the decision model for VOF.

Variability Design

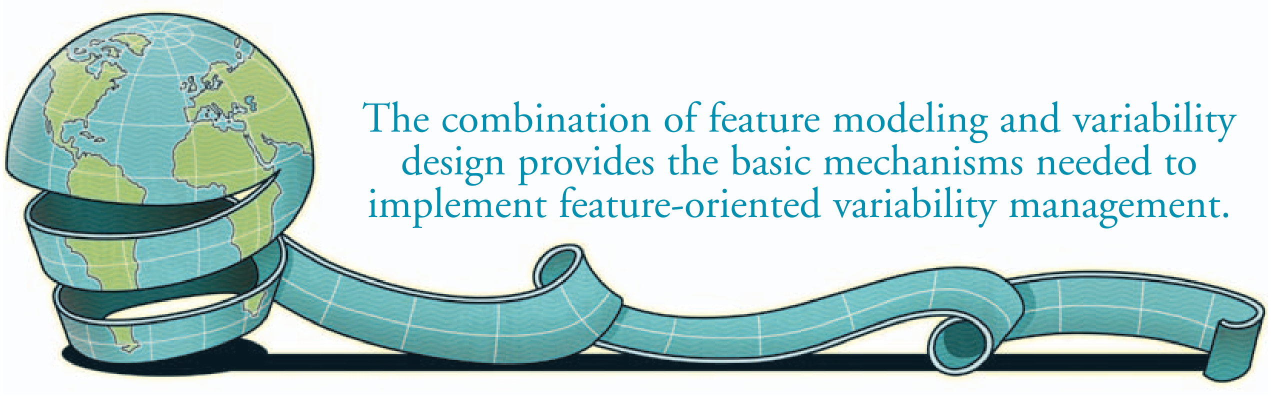
Variability design is the activity of constructing and evolving a reuse infrastructure, which realizes all variability requirements specified in a feature model. The major concept for designing variability is to encapsulate specified features (or related binding units) fully within single components at the architectural level. Optional units can thus later be easily included or excluded; additionally alternative realizations of the same but varying feature can conceptually be hidden from other related architectural elements.

In our example, an optional component Virtual Print Service Manager may be defined to encapsulate all functionalities related to the optional Virtual Printer feature in the feature model. As components are recursively refined and decomposed, variant concepts and components may also occur at all levels. A File Convert Handler realizing the File Converter feature may, for instance, be designed as an optional subcomponent of the optional Virtual Print Service Manager component.

Each architectural view or design artifact is potentially affected by variability and thus must be prepared to capture these product line concepts explicitly. Hence, design is captured analogously as in single system development except for extensions by variant model elements [4]. The structural view in Figure 3, for instance, depicts an excerpt of generic component decomposition for the VOF product line containing some variant parts. (Note that the black circles in Figure 3 represent variation points, at which selections of variants are controlled.)

In simple cases, decisions on the exclusion of individual variant elements can be taken independently of each other. It must, however, also be ensured that excluding model elements will result again in well-formed design models. From this perspective, excluding the Virtual Print Service Manager component also requires the exclusion of all connected components (for example, File Convert Handler). Also, there are typically other conceptual dependencies among variant elements. This is the case because some variability can, in general, not be encapsulated by a single element. For example, excluding the Virtual Print Service Manager component but including the Printer Registration Handler component is not a useful configuration of a product.

In addition to relationships and constraints among elements in a single model, decisions on variant features must be propagated consistently throughout all artifacts. Therefore, architecture and design documents must be accompanied by a deci-



The combination of feature modeling and variability design provides the basic mechanisms needed to implement feature-oriented variability management.

sion model [10], which explicitly captures all constraints among variant elements of all models in a table format. (See the table here for a part of the decision model for VOF.) A decision model is thus the analogous artifact in the solution space of a feature model for managing features and their interrelationships in the problem space as introduced earlier: variant model elements correspond to features; sets of model elements referring to the same varying feature correspond to binding units.

Eventually, the decision model links features in the problem space via a constraint network with variant design elements. As a result, variability can be traced from problem to solution space and vice versa. This combination of feature modeling and variability design thus establishes a systematic and consistent management of product line variability across all phases of the product line life cycle.

More complete technical details on the approach described here for feature modeling, and designing product lines and their variability from architecture to component implementation, are presented in [7] and [10].

Conclusion

This article has described techniques that enable the explicit modeling and management of product line variability across the life cycle from problem to solution space. The combination of feature modeling and variability design provides the basic mechanisms needed to implement feature-oriented variability management.

We have realized the presented approach several times in diverse industry contexts. Its implementations are already practically useful, however, we also uncovered several technical challenges that clearly require further research. Our approach must be extended to address other issues such as a formal base for analyzing consistency between various artifacts (including behavior specifications and architecture models) and a support for controlling and evolving variability information consistently along the evolution of a product line infrastructure. Most notably, managing and modeling varying non-functional or quality properties of products in a product line is not fully possible with available technologies. C

References

1. Batory, D. Feature models, grammars, and propositional formulas. In H. Obbink and P. Klaus, Eds., *Software Product Lines*. LNCS 3714. Springer-Verlag, Berlin Heidelberg, 2005, 7–20.
2. Czarnecki, K. and Eisenecker, U. *Generative Programming: Methods, Tools, and Applications*. Addison-Wesley, New York, 2000.
3. Ferber, S. et al. Feature interaction and dependencies: Modeling features for reengineering a legacy product line. In G.J. Chastek, Ed., *Software Product Lines*. LNCS 2319. Springer-Verlag, Berlin Heidelberg, 2005, 235–256.
4. Gomaa, H. and Saleh, M. Feature-driven dynamic customization of software product lines. In M. Morisio, Ed., *Reuse of Off-the-Shelf Components*. LNCS 4039. Springer-Verlag, Berlin Heidelberg, 2006, 58–72.
5. Jacobson, I., Griss, M., and Jonsson, P. *Software Re-use: Architecture, Process and Organization for Business Success*. Addison-Wesley, New York, 1997.
6. Kang, K., Cohen, S., Hess, J., Nowak, W., and Peterson, S. Feature-oriented domain analysis (FODA) feasibility study. Technical Report CMU/SEI-90-TR-21, Pittsburgh, PA, Software Engineering Institute, Carnegie Mellon University, 1990.
7. Kang, K., Lee, J., and Donohoe, P. Feature-oriented product line engineering. *IEEE Software 19*, 4 (July/August 2002), 58–65.
8. Lee, K. et al. Combining feature-oriented analysis and aspect-oriented programming for product line asset development. In *Proceedings of 10th International Software Product Line Conference (SPLC 2006)*, Baltimore, MD, 2006, 103–112.
9. Lee, J. and Kang, K. Feature binding analysis for product line component development. In F. van der Linden, Ed., *Software Product Family Engineering*. LNCS 3014. Springer-Verlag, Berlin Heidelberg, 2004, 266–276.
10. Muthig, D. and Atkinson, C. Model-driven product line architectures. In G.J. Chastek, Ed. *Software Product Lines*. LNCS 2379. Springer-Verlag, Berlin Heidelberg, 2002, 110–129.
11. Simos, M. et al. *Software Technology for Adaptable Reliable Systems (STARS) Organization Domain Modeling (ODM) Guidebook Version 2.0*. STARS-VC-A025/001/00, Lockheed Martin Tactical Defense Systems, Manassas, VA, 1996.

Jaejoon Lee (jaejoon.lee@iese.fraunhofer.de) is a scientist in the Product Line Architectures department at the Fraunhofer Institute for Experimental Software Engineering (IESE) in Kaiserslautern, Germany.
Dirk Muthig (dirk.muthig@iese.fraunhofer.de) is the department head of the Product Line Architecures department at the Fraunhofer Institute for Experimental Software Engineering (IESE) in Kaiserslautern, Germany.

This research is partially carried out in the Cluster of Excellence Dependable Adaptive Systems and Mathematical Modeling project, which is funded by the Program Wissen schafft Zukunft of the Ministry of Science, Education, Research and Culture of Rhineland-Palatinate, Germany, AZ.: 15212-52 309-2/40 (30).