Concepts and Guidelines of Feature Modeling for Product Line Software Engineering

Kwanwoo Lee, Kyo C. Kang, and Jaejoon Lee

Department of Computer Science and Engineering,
Pohang University of Science and Technology,
San 31 Hyoja-Dong, Pohang, 790-784, Korea
{kwlee, kck, gibman}@postech.ac.kr
http://selab.postech.ac.kr/index.html

Abstract. Product line software engineering (PLSE) is an emerging software engineering paradigm, which guides organizations toward the development of products from core assets rather than the development of products one by one from scratch. In order to develop highly reusable core assets, PLSE must have the ability to exploit commonality and manage variability among products from a domain perspective. Feature modeling is one of the most popular domain analysis techniques, which analyzes commonality and variability in a domain to develop highly reusable core assets for a product line. Various attempts have been made to extend and apply it to the development of software product lines. However, feature modeling can be difficult and time-consuming without a precise understanding of the goals of feature modeling and the aid of practical guidelines. In this paper, we clarify the concept of features and the goals of feature modeling, and provide practical guidelines for successful product line software engineering. The authors have extensively used feature modeling in several industrial product line projects and the guidelines described in this paper are based on these experiences.

1 Introduction

Product line software engineering (PLSE) is an emerging software engineering paradigm, which guides organizations toward the development of products from core assets rather than the development of products one by one from scratch. Two major activities of PLSE are core asset development (i.e., product line engineering) and product development (i.e., product engineering) using the core assets. (Details of the product line software engineering framework can be found at [1].)

The paradigm of developing core assets for application development has been called domain engineering (DE), in which emphasis is given to the identification and development of reusable assets from an application "domain" perspective. Product line software engineering is similar to domain engineering in that they both attempt to exploit commonalities to build reusable core assets. However, PLSE differs from DE in that PLSE is founded on marketing. In PLSE, a product plan that specifies target products and their features from a market analysis is the primary input. Fielding prod-

ucts with features that the market demands in a timely manner and then evolving those products as the market evolves is the major driving force in the asset development in PLSE. Therefore, the scope of analysis and development in PLSE can be narrower and more focused than in DE. However, most engineering techniques used in DE can be applied in PLSE as both paradigms attempt to build flexibility and reusability into core assets.

In order to develop reusable core assets, PLSE must have an ability to exploit commonality and manage variability. Although we build core assets for a product line, we must construct them with an understanding of the domain, which provides a wider engineering perspective for reusability and adaptability than a product line. Suppose, for example, a company manufactures freight elevators which aim at the factory market segment. Even if the immediate goal of product line engineering is to develop core assets for a set of freight elevators in the marketing plan, an understanding of the elevator domain, which may include passenger elevators, can improve the flexibility of core assets for future evolution. Therefore, domain analysis, which identifies commonality and variability from a domain perspective, is a key requirement for reusable core asset development for product lines.

Since Draco [2] was developed by Neighbors twenty yeas ago, various domain analysis techniques [2], [3], [4], [5], [6], [7], [8] have been proposed and developed. Neighbors coined the term “domain analysis” and defined it as the activity of identifying objects and operations of a class of similar systems in a particular problem domain. Many have attempted to develop domain analysis techniques, including Prieto-Diaz et al. at the Software Productivity Consortium and Kang et al. at the Software Engineering Institute. The work by Prieto-Diaz et al. is characterized as “faceted classification” [4], which originated from library science and was extended to implement a reusable software library. The primary focus of the work by Kang et al. was to establish feature oriented domain analysis (FODA) [3], which identifies and classifies commonalities and differences in a domain in terms of “product features.” Feature analysis results are then used to develop reusable assets for the development of multiple products in the domain. Since the idea of feature analysis was first introduced in 1990, FODA has been used extensively for DE and PLSE both in industry (e.g., Italia Telecom [19], Northern Telecom [9], Hewlett Packard Company [11], [22], Robert Bosch GmbH [21], etc.) and academia (e.g., Software Engineering Institute [17], [18], Technical University of Ilmenau [16], Pohang University of Science and Technology [13], [14], [15], etc.).

Several attempts have also been made to extend FODA. For example, ODM (Organization Domain Modeling) [10] builds on both the faceted classification approach and FODA, generalizes the notion of “feature”, and provides comprehensive guidelines for domain modeling. The main focus of ODM lies in defining a core domain modeling process that is independent of a specific modeling representation. FeatuRSEB [11] extended RSEB (Reuse-Driven Software Engineering Business) [12], which is a reuse and object-oriented software engineering method based on the UML notations, with the feature model of FODA. The feature model of FeatuRSEB is used as a catalogue of or index to the commonality and variability captured in the RSEB models (e.g., use case and object models). The authors have also extended the original FODA to address the issues of reference architecture development [13], object-

oriented component engineering [14], and product line software engineering [15]. Recently, Czarnecki et al. developed DEMRAL [16], which is a domain engineering method for developing algorithmic libraries. A fundamental aspect of DEMRAL is feature modeling, which is used to derive implementation components for a product line and configure components in a product line.

All such feature-oriented methods are still evolving and being applied to various application domains. Some examples are listed below:

- The Army Movement Control Domain [17]
- The Automated Prompt Response System Domain [18]
- The Electronic Bulletin Board Systems Domain [13]
- The Telephony Domain [19]
- The Private Branch Exchange Systems Domain [20]
- The Car Periphery Supervision Domain [21]
- The Elevator Control Software Domain [14, 15]
- The E-Commerce Agents Systems Domain [22]
- The Algorithmic Library Domain [16]

There are several reasons why feature-oriented domain analysis has been used extensively compared to other domain analysis techniques. The first reason is that feature is an effective communication "medium" among different stakeholders. It is often the case that customers and engineers speak of product characteristics in terms of "features the product has and/or delivers." They communicate requirements or functions in terms of features, and such features are distinctively identifiable functional abstractions to be implemented, tested, delivered, and maintained. We believe that features are essential abstractions that both customers and developers understand, and should be first class objects in software development. The second reason is that feature-oriented domain analysis is an effective way to identify variability (and commonality) among different products in a domain. It is natural and intuitive to express variability in terms of features. When we say "the features of a specific product," we use the term "features" as distinctive characteristics or properties of a product that differ from others or from earlier versions (i.e. new features). The last reason is that the feature model can provide a basis for developing, parameterizing, and configuring various reusable assets (e.g., domain requirement models, reference architectural models, and reusable code components). In other words, the model plays a central role, not only in the development of the reusable assets, but also in the management and configuration of multiple products in a domain.

Feature modeling is considered a prerequisite for PLSE, and it is gaining popularity among practitioners and researchers. However, most users of feature modeling have difficulty in applying it to product line engineering. This difficulty comes from the imprecise understanding of feature models and the lack of practical guidelines for modeling. Thus, in this paper, we strive to clarify what feature model is and how it is used, and provide practical guidelines for the successful PLSE. The authors have extensively used feature modeling in several industrial problems and the guidelines in this paper are based on these experiences.

2 What Is Feature?

Informally, features are key distinctive characteristics of a product. We see different domain analysis methods use the term "feature" with slightly different meanings. FODA defines a feature as a prominent and distinctive user visible characteristic of a system. Several methods [11, 13] have been developed and extended based on this definition. ODM generalized the definition in FODA and defined it as a distinguishable characteristic of a "concept" (e.g., artifact, area of knowledge, etc.) that is relevant to some stakeholders (e.g., analysts, designers, and developers) [10]. Unlike FODA, ODM focuses on modeling key concepts that represent semantics of a domain. Based on the concepts, features are modeled as differentiators of concepts or multiple instances of a concept in the domain. DEMRAL [16] and Capture [5] follow the notion of feature in ODM.

Our notion of feature follows the FODA definition. Features are any prominent and distinctive concepts or characteristics that are visible to various stakeholders. Unlike ODM, we do not explicitly distinguish a concept from a feature. This is because the distinction between a concept and a feature is often unclear in certain situations. Consider a simple example of an elevator domain. Indicating the position of an elevator cage is a key operational concept in an elevator domain. Most elevators indicate their positions through lanterns while others use voice indication systems. In this example, it seems natural to model *lantern* and *voice* as presentation features of the operational concept (i.e., indicating a cage's position). However, if we become interested in lighting devices, the lantern can be modeled as a concept. Therefore, too much emphasis on the distinction between a concept and a feature may impose a burden on domain analysts. It is important to note that the focus of feature modeling should be laid in identifying commonality and variability in a domain rather than differentiating concepts from features. On the other hand, the fuzzy nature of feature makes it difficult to formalize its precise semantic, validate results, and provide automated support [23].

What are the differences between a feature and other conceptual abstractions (i.e., function, object, and aspect)? Functions, objects, and aspects have been mainly used to specify the internal details of a system. Structured methods specify internal details of a system in terms of functions, which represent procedural abstractions; object-oriented methods specify the structure and behavior of a system in terms of objects, which are uniquely identifiable key abstractions or entities. Recently, Aspect-Oriented Programming (AOP) postulates other abstractions crosscutting the boundary of modular units (e.g., functions and objects). AOP defines these crosscutting abstractions as *aspects* and specifies them separately from other modular units (e.g., objects). In other words, functions, objects, and aspects are conceptual abstractions that are identifiable from internal viewpoints. On the other hand, features are externally visible characteristics that can differentiate one product from the others.

Therefore, feature modeling must focus on identifying external visible characteristics of products in terms of commonality and variability, rather than describing all details of products such as other modeling techniques (e.g., functional modeling, object-oriented modeling, etc.). From the understanding of commonality and variability of products, we can derive functions, objects, aspects, etc. in a reusable form.

3 Feature Modeling Overview

Feature modeling is the activity of identifying externally visible characteristics of products in a domain and organizing them into a model called a feature model. The feature modeling described in this section is based on that of FORM [13].

Product features are identified and classified in terms of capabilities, domain technologies, implementation techniques, and operating environments. Capabilities are user visible characteristics that can be identified as distinct services (e.g., call forwarding in the telephony domain), operations (e.g., dialing in the telephony domain), and non-functional characteristics (e.g., performance). Domain technologies (e.g., navigation methods in the avionics domain) represent the way of implementing services or operations. Implementation techniques (e.g., synchronization mechanisms) are generic functions or techniques that are used to implement services, operations, and domain functions. Operating environments (e.g., operating systems) represents environments in which applications are used.

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 domain 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, captures structural or conceptual relationships among features. Three types of relationships are represented in this diagram. The *composed-of* relationship is used if there is a whole-part relationship between a feature and its sub-features. In cases where features are generalization of sub-features, they are organized using the *generalization/ specialization* relationship. The *implemented-by* relationship is used when a feature (i.e., a feature of an implementation technique) is necessary to implement the other feature.

Composition rules supplement the feature model with mutual dependency and mutual exclusion relationships which are used to constrain the selection from optional or alternative features. That is, it is possible to specify which features should be selected along with a designated one and which features should not.

Consider the Private Branch Exchange (PBX) systems domain as an example of feature modeling. PBX is a telephone system that switches calls between users within a small area, such as a hotel, an office building, or a hospital. Such switching is the primary service of PBX. In addition, it provides many supplementary services such as 'call forwarding' and 'call back'.

Fig. 1 shows a simplified feature model of PBX. Capability features of PBX consist of service features (e.g., *Call Request*, *Call Forwarding*, and *Call Answer*), operation features (e.g., *Tone Operation*, *LCD Control*, *Dial*, etc.), and a non-functional feature (e.g., *BHCA*[1]). Operating environment features of PBX are *Single Line*, *Multifunction*, and *ISDN*, each of which represents a different type of telephone terminal.

Domain technology features represent the way of implementing services or operations. In the PBX domain, *Digit Analysis* [2]and *Switching Method* [3]are used to imple-

[1] BHCA (Busy Hour Call Attempt) indicates the number of call attempts that PBX can handle for an hour.

[2] Digit Analysis is a method for analyzing digits of a requested call to determine its destination.

ment services of PBX. Compared to domain technology features, implementation technique features are more generic and might be applicable to other domains. In this example, the *Search Algorithm* feature is used to implement a domain method (i.e., Digit Analysis), but this can be used in other domains (e.g., a graph library domain).

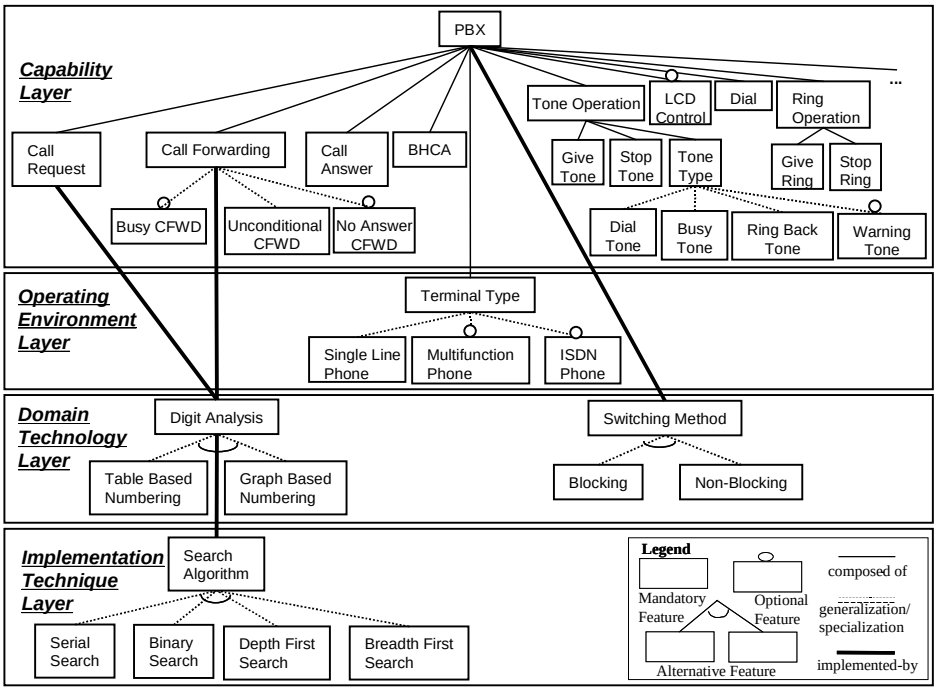

Fig. 1. A feature model example: PBX product line

All products in the PBX domain have common services such as *Call Request*, *Call Forwarding*, and *Call Answer*, which are modeled as mandatory features. On the other hand, services such as *Busy CFWD*, *No Answer CFWD*, and *Warning Tone* may not exist in a certain product, so they are modeled as optional features. Since only one of the two digit analysis methods (e.g., *Table Based Numbering* and *Graph Based Numbering*) can be used for a product, the features are modeled as alternative features.

The *PBX* feature consists of service, operation, and operating environment features, so they are organized with the *PBX* feature through the *composed-of* relationship. However, since *Digit Analysis* is used to implement *Call Request* and *Call Forwarding*, the *implemented-by* relationship is used among them. *Busy CFWD*, *Unconditional CFWD*, and *No Answer CFWD* are specializations of *Call Forwarding*, so they are organized according to the generalization/specialization relationship. The composition

[3] Switching Method indicates the way of connecting a caller to a callee.

rules further constrain feature selection. For example, the lower portion of Fig. 1 shows that the *Table Based Numbering* feature must be selected along with the *Serial Search* or *Binary Search* feature, and *Multifunction Phone* and *ISDN Phone* must be selected with *LCD Control*.

4 Guidelines for Feature Modeling

Understanding commonality and variability from a domain perspective is essential for developing reusable core assets for a product line. Feature modeling is an effective and efficient method for identifying and organizing commonality and variability from a domain perspective.

At first glance, feature modeling seems to be intuitive. However, it is difficult to build a good feature model, which makes it easy to understand the problem domain in terms of commonality and variability and assists the development of reusable assets. We believe that the guidelines described in this paper will be helpful for those who are novices in domain analysis, especially feature analysis. We have applied feature modeling in several industrial cases and the following guidelines have been developed based on our work.

4.1 Domain Planning Guidelines

Domain planning is a preliminary phase to complete before starting feature modeling. This consists of activities for selecting a domain, clarifying the boundary of the selected domain, organizing a domain analysis team, and making a domain dictionary.

Domain Selection. The first step for domain planning is to determine the right domain (i.e., a set of products) that leads to successful product line engineering.

- *For organizations that do not have any domain engineering experience, we recommend that a domain be selected that is small and has a high level of commonality among the products*. Domain engineering, in general, requires a large initial investment and a long development time compared to single product development. Selecting a reasonable size for a domain with much commonality among products is the first step toward the economical and successful product line software engineering. After learning and exploring with the first domain, organizations should consider adding more and more product lines to the domain.
- *Exclude products with fixed delivery schedules from a domain*. In case a domain includes products with fixed delivery schedules, the manager will assign a higher priority to the development of the products instead of performing a detailed domain analysis as the time to deliver the products draws near. So products with fixed delivery schedules must be excluded from a target domain for successful feature modeling.

Domain Scoping. Once a right domain is selected, domain analysts should determine the boundary of a domain and relationships between the domain elements and other entities outside the domain, and should share information with each other. Without the consensus of the domain boundary, time-consuming discussions among domain modelers are unavoidable.

- *If there is a high degree of contextual differences, the domain may need to be re-scoped to a narrower domain.* Understanding of the extent of contextual differences (e.g., differences in underlying hardware) will help to re-scope the domain. For example, an elevator domain has *parking building* and *freight* elevator product lines. The variations in the underlying hardware (e.g., different types of devices) between them are high. This makes it very difficult to abstract out common hardware abstractions. In this case the domain must be re-scoped to a narrower domain, which includes only one of *parking building* and *freight* elevator product lines. Thus, the re-scoped domain is much easier to build common hardware interface components, which can handle different types of devices without major modification.

A Domain Analyst Team Organization. The purpose of domain modeling is to capture common and different concepts or characteristics in system development. Therefore, different stakeholders must be included in the domain analyst team. Different viewpoints from different stakeholders can be helpful for identifying important domain concepts or capabilities. For example, end-users help domain analysts identify services or functions provided by the products in a domain; domain experts might be helpful for identifying domain technologies (e.g., navigation methods in the avionics domain) that are commonly used in a domain; and developers provides generic implementation techniques.

- *Experts in different product lines of the same domain should be involved in a domain analysis activity.* No single individual has sufficient experience concerning all product-lines within the same domain. Domain experts from different product lines can provide a wider engineering perspective for reusability and adaptability. Therefore, in order to build highly reusable core assets, domain analysis must be performed with many domain experts from different product lines.

A Domain Dictionary. After organizing a domain analysis team, the last activity for domain planning is to prepare a domain dictionary for feature modeling.

- *In an immature or emerging domain, standardize domain terminology and build a domain dictionary.* In an immature or emerging domain, different terminologies with the same meaning may be used for different product lines. Standardizing domain terminologies is an essential activity in preparation for feature modeling. If not done, different perceptions of domain concepts could cause confusion among participants in modeling activities and lead to time-consuming discussions.

Domain planning is an important first step toward successful feature modeling. Without understanding the exact scope of the domain, the intended use of domain

products, various external conditions, and common domain vocabularies, feature identification can be very difficult. Even the identified features may be useless. The next section describes the guidelines for feature identification.

4.2 Feature Identification Guidelines

Identification of features involves abstracting domain knowledge obtained from the domain experts and other documents such as books, user manuals, design documents, and source programs. The volume of documents to be analyzed, however, tends to be enormous in a domain of any reasonable size. In this context, the following guideline is useful.

- *Analyze terminologies used in the product line to identify features*. We often see that domain experts in mature or stable domains use standard terms to communicate their idea, needs, and problems. For example, 'call-forwarding' and 'call-back' represent standard services in the telephony domain. Using standard terms for the feature identification can expedite communication between domain analysts and information providers (e.g., domain experts and end-users). Our experience has shown that analyzing standard terminologies is an efficient and effective way to identify features.

In our method, we provide feature categories, shown in Fig. 2, as a feature identification framework. The feature categories may be incomplete and not cover the entire domain knowledge space, but we are confident that these categories are the first applicable guidelines for identifying product features.

As discussed earlier, capability features are characterized as distinct services, operations, or non-functional aspects of products in a domain. Services are end-user visible functionality of products offered to their users in order to satisfy their requirements. They are often considered as marketable units or units of increment in a product line. Operations are internal functions of products that are needed to provide services. Non-functional features include end-user visible application characteristics that cannot be identified in terms of services or operations, such as presentation, capacity, quality attribute, usage, cost, etc.

There may be several ways to implement services or operations. Domain technology features are domain specific technologies that domain analysts or architects use to model specific problems in a domain or to "implement" service or operation features. Domain-specific theories and methods, analysis patterns, and standards and recommendations are examples. Note that these features are specific to a given domain and may not be useful in other domains.

Implementation technique features are more generic than domain technology features and may be used in different domains. They contain key design or implementation decisions that may be used to implement other features (i.e., capability and domain technology features). Communication methods, design patterns, architectural styles, ADTs, and synchronization methods are examples of implementation techniques.

Applications may run in different operating environments. They may run on different hardware or operating systems, or interface with different types of devices or ex-

ternal systems. Operating environment features include product line contexts, such as computing environments and interfaces with different types of devices and external systems. Protocols used to communicate with external systems are also classified as environment features.

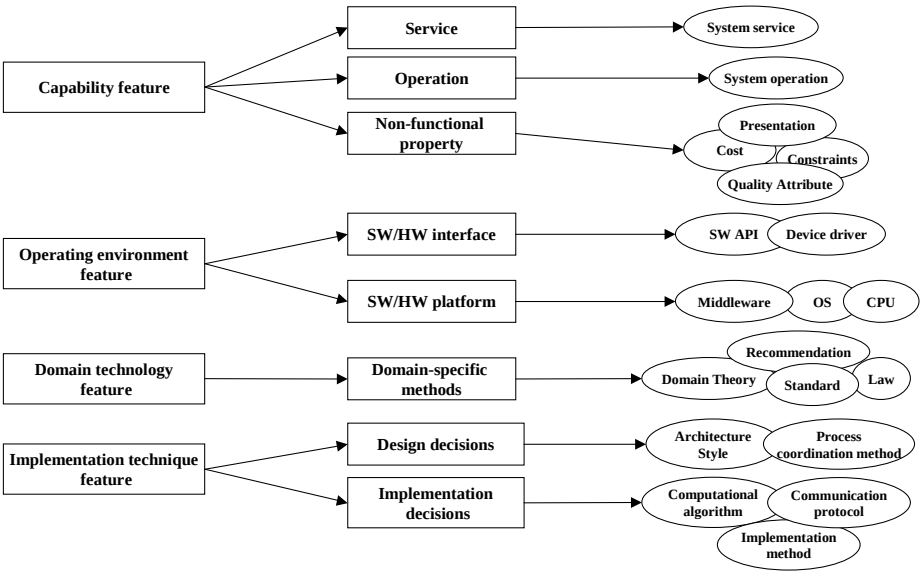

Fig. 2. Feature categories

Although the above feature categories are useful guidelines for identifying features, many people often make common mistakes in identifying features. The following guidelines must always be kept in mind during feature identification activity.

- *Try to first find differences among products envisioned for a product line and then, with this understanding, identify commonalities.* Products in the same product line share a high level of commonality. Hence, the “commonality space” would be larger to work with than the "difference space." It is our experience that finding differences is much easier than finding commonalities. We recommend, first, to identify product categories (e.g., freight and passenger elevators in an elevator product line), within each of which products should have a higher level of commonality and lower variability than those in other categories. Next, list features that characterize each category, i.e., differences between categories. Then, for each category, list products and do the same thing. With this understanding, we experienced that identification of commonalities became easier and we could proceed to listing common features and modeling them effectively and efficiently.
- *Do not identify all implementation details that do not distinguish between products in a domain.* A skillful developer tends to enumerate all the implementation details and identify them as features, even though there are no variations among them. It is important to note that a feature model is not a requirement model, which expresses the details of internal functions. The modeling focus should be

on identifying properties, factors, assumptions that can differentiate one product from others in the same domain, not on finding all implementation details that are necessary to implement the products in a domain.

Once candidate features are identified based on the above guidelines, these features should be organized into a feature model. The following guidelines show how to organize a set of features into a feature model that is easy to understand and helpful for the development of reusable assets.

4.3 Feature Organization Guidelines

As noted earlier, features are organized into a feature diagram in terms of *composed-of*, *generalization/specialization*, and *implemented-by* relationships. Further, composition rules are used to specify mutual inclusive and exclusive relationships between variant features (i.e., optional and alternative features) that do not have an explicit relationship in a feature diagram.

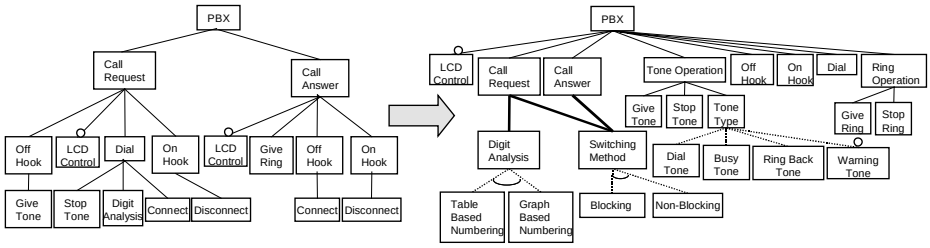

Fig. 3. A feature diagram like a functional call hierarchy

– *Do not organize features to represent functional dependencies, like a function call hierarchy, but organize features to capture and represent commonalities and differences.* Developers who are familiar with a structured development method often confuse a feature diagram with a functional call hierarchy. Thus they tend to identify all functions of products as features and organize them similar to a functional call hierarchy. However, features should be organized so that commonalities and variabilities can be recognized easily rather than representing interactions among features, like a function call hierarchy. For example, the left feature diagram of Fig. 3 merely enumerates all operations related to Call Request and Call Answer and organizes them like a functional call hierarchy. However, the right feature diagram of Fig. 3 shows common services (e.g., Call Request), operations (e.g., Tone Operation), and domain technologies (e.g., Switching Method) and explains how they are differentiated among products in a domain. For instance, Connect and Disconnect in the left feature diagram are grouped into a common domain method, i.e., Switching Method, which is further refined into Blocking and Non-Blocking to express variability in a domain.

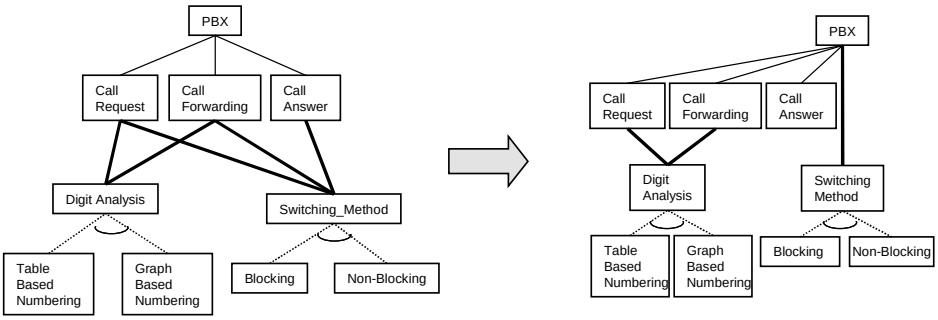

Fig. 4. An example of reducing complexity of a feature model

– *If many upper-level features are associated with a lower-level feature through the implemented-by relationship, reduce complexity of the feature model by associating the lower-level feature with the nearest common parent of its related upper-level features.* When upper-level features in a feature diagram are associated with several lower-level features through the implementation-by relationship, their implementation relationships between those features may be very complex. For example, the left feature diagram in Fig. 4 shows complex relationships between three service features (i.e., *Call Request*, *Call Forwarding*, and *Call Answer*) and two domain technology features (i.e., *Digit Analysis* and *Switching Method*). Since the primary concern of feature modeling is to represent commonality and variability in a domain rather than to model implementation relationships between features, complex relationships can be reduced by associating the lower-level feature with the nearest common parent of its related upper-level features. In this example, complexity is reduced by associating the *Switching Method* with *PBX*, which is the nearest common parent of its related service features (i.e., *Call Request*, *Call Forwarding*, and *Call Answer*). (See the right feature model in Fig. 4.)

4.4 Feature Refinement Guidelines

The feature model should be reviewed with domain experts who did not participate in the feature identification activity and with domain products, such as manual, programs, design documents, systems, etc. During the reviewing process, if necessary, a set of feature may be added, deleted, or refined.

– *Refine the feature model so that the logical boundary created by a feature model can correspond to the physical boundary created by its architecture models.* Product-line software should be adaptable for a set of features selected for a specific application. By designing each selectable feature as a separate component, applications can be derived easily from product-line software. If there is difficulty in establishing this relation, the feature must be refined into specific features so that features can be easily mapped into architectural components. This will enhance the traceability between architectural components and features. For example, the *Motor Control* feature was originally allocated to a single subsys-

tem, *MotorControlSubsystem*. However, a certain type of elevator products (e.g., high-speed passenger elevators) requires a precise position control within its timing requirement. Thus we had to allocate sub-functions (i.e., position control) of the *Motor Control* feature into the different subsystem to satisfy the timing requirement. To enhance the traceability between architectural components and their features, we refined the *Motor Control* feature into *Position Control* and *Moving Control* features, as shown in Fig. 5.

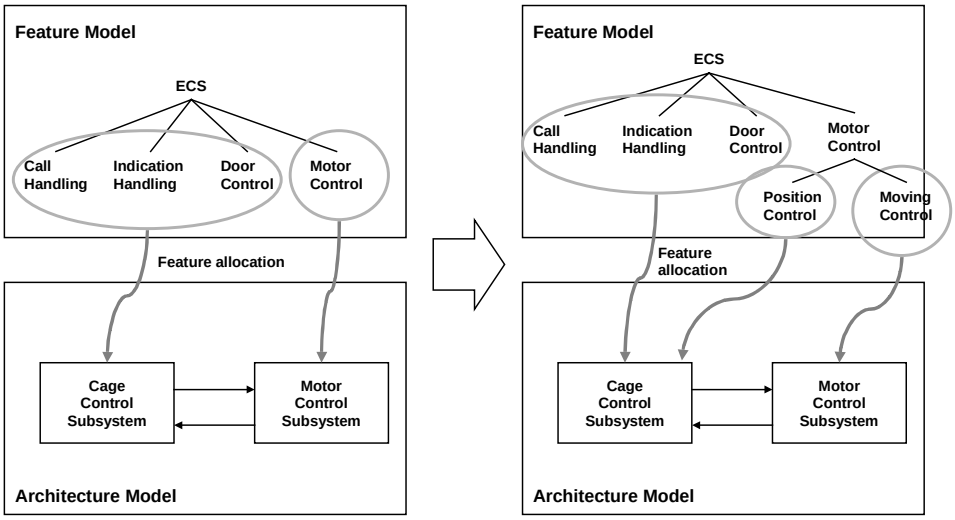

Fig. 5. An example of refinement of a feature whose sub-features are allocated into different architectural components

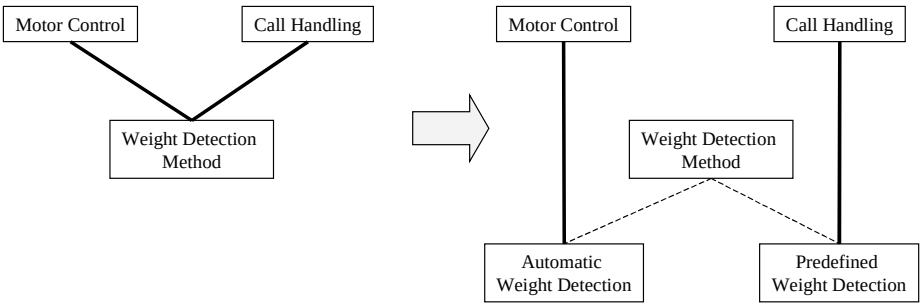

Fig. 6. An example of refinement of a common feature (e.g., Weight Detection Method)

– *Refine an abstract feature until no variations are exposed among products in a domain.* A feature need not be further refined into sub features if there are no variations among different products. However, although no variation exists in a feature among different products of a domain, if its refined sub features are associated with other features in the upper level, they may be refined further. For example, in the elevator control software domain [14], [15], the *Weight Detection*

Method is an abstract domain method that is necessary to implement various operations (e.g., *Motor Control*, *Call Handling*, etc.). However, this feature is used somewhat differently according to different high-level features, for example, *Motor Control* mainly uses *Automatic Weight Detection* and *Call Handling* uses *Predefined Weight Setting*, the *Weight Detection Method* can be further refined, as shown in Fig. 6.

During the refinement of features, can any differences among products in a domain be a feature? For example, if two different developers implement the same algorithm with different programming styles, can the programming styles be identified as features? If the window size of a product is somewhat different from others, can the window size be a feature? In our view, features should be essential characteristics or properties in a domain. Therefore, if the differences are not essential in a domain, those differences can be eliminated through a standardization process. However, if they have special meanings – one programming style supports for understandability of a program and the other style for efficiency, they can be identified as features.

Conclusion

We have applied our extensive experience with feature modeling to four domains, which include the electronic bulletin board systems (EBBS) [13], a private branch exchange (PBX) [20], elevator control software (ECS) [14], [15], and slab yard inventory systems domains [25]. We have found that feature modeling is intuitive, efficient, and effective for identifying commonality and variability of applications in a domain. The feature model is a good communication media among those who have different concerns in application development.

The following are lessons learned from our experience.

- An organization with a standard infrastructure (e.g., physical communication media) for its product line is usually unwilling to analyze the anticipated changes of the external environments or the implementation techniques. Engineers in such an organization tend to believe that there are few variations in terms of operating environments or implementation techniques as the products have been built on the standard of the organization. Therefore, we convinced them to analyze future changes as well as current variations, since the standard could also be changed.
- In an unstable domain, standardizing domain terminology and using the standard terms during analysis could expedite the feature identification process. Without standard terms, the team suffered greatly from time-consuming discussions on minor semantic differences between features. Therefore, analyzing the standard terminology is an effective and efficient way to identify features for the domain
- In an emerging domain (e.g., a cellular phone domain), capability features, such as services or operations, change more frequently than domain technology and implementation technique features. On the other hand, in a stable domain (e.g., an elevator domain), services are nearly changed but operating environments and implementation techniques are changed more often than services.

Tool support for feature modeling is indispensable to handle hundreds of product features in a product line. We have developed a feature modeling tool, which can handle many product features (e.g., more than five hundred product features) efficiently through sub-tree expansion and hiding capabilities and check the consistency of composition rules in the feature model. It runs on any PC or workstation on which the Java development kit (JDK) software is installed for its run-time environment. The tool is available upon request via kck@postech.ac.kr.

References

1. Clements, P., Northrop, L.: Software Product Lines: Practices and Patterns, Addison-Wesley, Upper Saddle River, NJ (2002)
2. Neighbors, J.: The Draco Approach to Construction Software from Reusable Components, *IEEE Transactions on Software Engineering* SE-10, 5 (1984) 564-573
3. Kang, K., Cohen, S., Hess, J., Nowak, W., 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)
4. Prieto-Diaz, R.: Implementing Faceted Classification for Software Reuse, *Communications of the ACM* 34, 5 (1991) 88-97
5. Bailin, S.: Domain Analysis with KAPTUR, *Tutorials of TRI-Ada'93*, New York, NY (1993)
6. Frakes, W., Prieto-Diaz, R., Fox, C.: DARE: Domain Analysis and Reuse Environment, Annals of Software Engineering 5 (1998) 125-151
7. Coplien, J., Hoffman, D., Weiss, D.: Commonality and Variability in Software Engineering, *IEEE Software* 15, 6 (1998) 37-45
8. Weiss, D. M., Lai, C. T. R.: Software Product-Line Engineering: A Family Based Software Development Process, Addison-Wesley, Reading, MA (1999)
9. Zalman, N. S.: Making the Method Fit: An Industrial Experience in Adopting FODA, *Proc. Fourth International Conference on Software Reuse*, Los Alamitos, CA (1996) 233-235
10. Simos, M. et al, Software Technology for Adaptable Reliable Systems (STARS) Organization Domain Modeling (ODM) Guidebook Version 2.0, *STARS-VC-A025/001/00*, Manassas, VA, Lockheed Martin Tactical Defense Systems (1996)
11. Griss, M. L., Favaro, J., d'Alessandro, M.: Integrating Feature Modeling with the RSEB, *Proc. Fifth International Conference on Software Reuse*, Victoria, BC, Canada (1998) 76-85
12. Jacobson, I., Griss, M., Jonsson, P.: *Software Reuse: Architecture, Process and Organization for Business Success*, Addison-Wesley, New York (1997)
13. Kang, K., Kim, S., Lee, J., Kim, K., Shin E., Huh, M.: FORM: A Feature-Oriented Reuse Method with Domain-Specific Reference Architectures, *Annals of Software Engineering*, 5 (1998) 143-168
14. Lee, K., Kang, K., Chae, W., Choi, B.: Feature-Based Approach to Object-Oriented Engineering of Applications for Reuse, *Software-Practice and Experience* 30, 9 (2000) 1025-1046
15. Lee, K., Kang, K., Koh, E., Chae, W., Kim, B., Choi, B.: Domain-Oriented Engineering of Elevator Control Software: A Product Line Practice, In Proceedings of the First Software Product Line Conference (SPLC), August 28-31, 2000, Denver, Colorado, USA,

Donohoe, P. (Eds.), *Software Product Lines: Experience and Research Directions*, Kluwer Academic Publishers, Norwell, Massachusetts (2000) 3-22
16. Czarnecki, K., Eisenecker, U., *Generative Programming: Methods, Tools, and Applications*, Addison-Wesley, New York (2000)
17. Cohen, S. G., Stanley Jr., J. L., Peterson, A. S., Krut Jr., R. W.: Application of Feature-Oriented Domain Analysis to the Army Movement Control Domain, *Technical Report, CMU/SEI-91-TR-28*, ADA 256590, Pittsburgh, PA, Software Engineering Institute, Carnegie Mellon University (1991)
18. Krut, R., Zalman, N.: Domain Analysis Workshop Report for the Automated Prompt Response System Domain, *Special Report, CMU/SEI-96-SR-001*, Pittsburgh, PA, Software Engineering Institute, Carnegie Mellon University (1996)
19. Vici, A. D., Argentieri, N.: FODAcom: An Experience with Domain Analysis in the Italian Telecom Industry, *Proc. Fifth International Conference on Software Reuse*, Victoria, BC, Canada (1998) 166-175
20. Kang, K., Kim, S., Lee, J., Lee, K.: Feature-Oriented Engineering of PBX Software for Adaptability and Reusability, *Software-Practice and Experience* 29, 10 (1999) 875-896
21. Hein, A., Schlick, M., Vinga-Martins, R.: Applying Feature Models in Industrial Settings, In Proceedings of the First Software Product Line Conference (SPLC), August 28-31, 2000, Denver, Colorado, USA, Donohoe, P. (Eds.), *Software Product Lines: Experience and Research Directions*, Kluwer Academic Publishers, Norwell, Massachusetts (2000) 47-70
22. Griss, M.: Implementing Product-Line Features by Composing Aspects, In Proceedings of the First Software Product Line Conference (SPLC), August 28-31, 2000, Denver, Colorado, USA, Patrick Donohoe (Eds.), *Software Product Lines: Experience and Research Directions*, Kluwer Academic Publishers, Norwell, Massachusetts (2000) 47-70
23. Simos, M., Anthony, J.: Weaving the Model Web: A Multi-Modeling Approach to Concepts and Features in Domain Engineering, *Proc. Fifth International Conference on Software Reuse*, Victoria, BC, Canada (1998) 166-175
24. Kiczales, G., Lamping, J., Mendheker, A., Maeda, C., Lopes, C., Loingtier, J., Irwin, J.: Aspect-Oriented Programming, In Proceedings of ECOOP'97 – Object-Oriented Programming, 11th European Conference, Jyvaskyla, Finland, June 1997, Aksit, M., Matsuoka, S. (Eds.), LNCS 1241, Springer-Verlag, Berlin and Heidelberg, Germany (1997)
25. Kang, K., Lee, K., Lee, J., Kim, S.: Feature Oriented Product Line Software Engineering: Principles and Guidelines, to appear as a chapter in *Domain Oriented Systems Development – Practices and Perspectives*, UK, Gordon Breach Science Publishers (2002)