Chapter 2
Variability Modeling

Kyo C. Kang and Hyesun Lee

What you will learn in this chapter
- *The different aspects and viewpoints of variability modeling one needs to consider in software product line engineering*
- *How these different viewpoints are interrelated to each other*
- *Variability modeling techniques*

1 Introduction

The aim of this chapter is to provide a comprehensive description of the notion of variability modeling in the context of software product line engineering and to give an overview of the techniques proposed for variability modeling.

Since its first introduction in 1990, feature modeling [1] has been the most popular technique to model commonality and variability (C&V) of products of a product line. Commonalities and variabilities are modeled from the perspective of *product features*, "stakeholder visible characteristics of products" in a product line that are of stakeholders' concern. For example, the fund transfer feature of a banking system may be of interest to customers, i.e., a service feature, but how the fund transfer happens may not be of interest to customers as long as it is done securely. However, it will be an important concern for the designer of the system and, when there are alternative ways, it is the responsibility of the designer to choose the right one for the target system.

The original feature model, FODA [1], is a simple model with features that are organized using "consists of" and "generalization/specialization" relationships

K.C. Kang (✉) • H. Lee
Pohang University of Science and Technology (POSTECH), Pohang, Republic of Korea
e-mail: kck@postech.ac.kr; compial@postech.ac.kr

using the AND/OR graph. Features are typed as mandatory, alternative, or optional features to represent C&V. Attributes of a feature may also be documented.

As it has gained a wide acceptance both by practitioners and researchers, this rather simple model was extended by many researchers introducing new modeling primitives such as feature cardinality and XOR relationships. Various research activities followed such as formal analysis of feature model, feature configuration, generative programming, etc. Also, there are a wide variety of product lines FODA and its extensions have been applied to, and it has been reported that C&V models tend to become complex as the size of product line increases. This complexity of a model is highly correlated with the complexity of the problem domain that is modeled. However, it has been noticed that many different types of C&V information, such as product goals as well as functional and design features, are all integrated into a single model which makes a C&V model even more complex.

In this section, we explore various dimensions of C&V in product line engineering. We separate C&V modeling into problem and solution space modeling. Problem space modeling is further refined to product goal, usage context, and quality attribute C&V modeling. Also, solution space modeling is refined to capability/service, operating environment, and design feature C&V modeling. Relationships/traceability between these models is managed separately from these models.

2 Concepts

The most important attribute of software is the “softness” of software, i.e., software that is easy (cost effective) to modify and adapt to evolving requirements or changing operating environments, easy to port on different hardware or software platforms, and easy to reuse for development of similar applications. Softness of software cannot be attained without engineering it into software. To embed softness into software, there have been many software engineering principles and concepts proposed, such as information hiding, program families, modularity, design patterns, etc.

In order to apply these design principles and concepts, however, we need to understand the commonality and variability (C&V) of the product line, i.e., a family of products. We need to explore the “space” of C&V of the products in a product line and potential evolution (“time”-dependent variability) of these products in the future, and then organize and codify the knowledge gathered as a C&V model. With this understanding of C&V, we can engineer software applying various design principles and embedding variation points that can later be bound with variants. For example, design decisions (design features) that can change may be encapsulated into software components applying the information hiding principle, and each changeable decision (alternative design features) can be implemented as a variant.

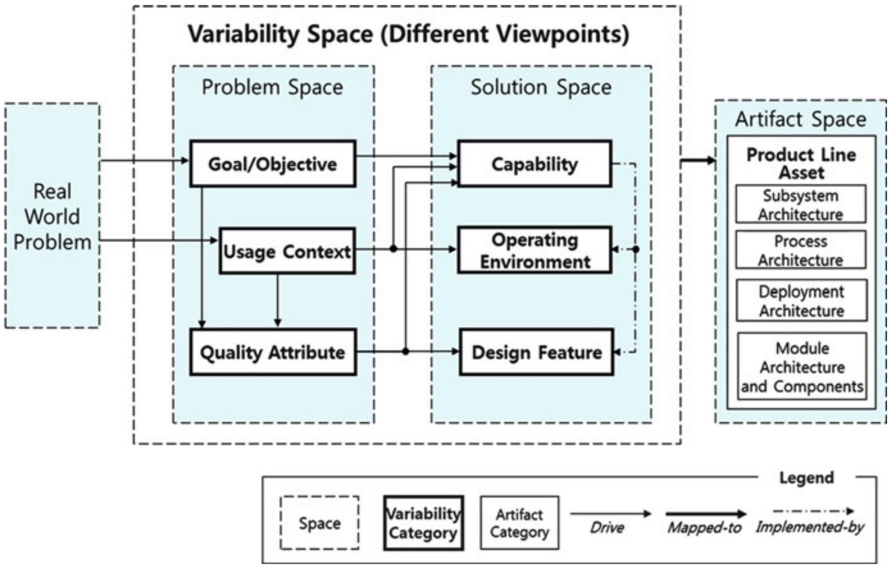

Fig. 2.1 Variability modeling space

In the following section, we explore different dimensions of variability of a software product line.

3 Commonality and Variability Modeling: The Scope

C&V of a product line can be modeled in many different ways based on different viewpoints (i.e., separation of different concerns). Largely, we can separate the problem space from the solution space[1] (see Fig. 2.1). For the problem space, user goals and objectives, required quality attributes, and product usage contexts are typically modeled in product line engineering. Within the solution space, C&V is typically modeled for the functional dimension (i.e., capabilities, services), the operating environmental dimension (e.g., operating systems, platform software, etc.), and the design dimension (e.g., domain technologies). C&V explored and modeled for these dimensions are materialized as software architectures, components, variation points, and variants in the artifact space. Implementation mechanisms such as inheritance, template, framework, macro, and generator may be used to implement variation points and variants.

[1] The terms "problem" and "solution" are relative. A solution for one may be a problem for others to solve. Requirements, which are considered "problems" to solve by designers, are "solutions" to real-world problems. One may view features in the "solution" space as problems for asset development in the artifact space.

The goals and objectives modeled for a product line defines “problems” at their highest level of abstraction to be addressed by the products of a product line, and therefore, they drive derivation of capabilities and quality attributes, which in turn may trigger derivation of other capabilities in the solution space. For example, the goal of moving passengers between floors safely will require elevator “capabilities” such as cabin moving, call handling, and door operation in addition to the obstacle detection for safety, a quality attribute. Techniques for implementing capability features are modeled as design features, each of which has associated quality attributes. For example, different obstacle detection devices may have different performance characteristics.

Typically, products used in different usage contexts require different capabilities and/or different quality attributes. For example, elevators in a hospital require a higher quality floor leveling feature than those in an office building to let wheel chairs and other medical equipment rolled in and out of an elevator easily. A flash memory for USB drivers needs a higher frequency data update than those built into a camera, for example, as they may be pulled out anytime. It should be noted that what derives decisions on quality requirements, operating environmental elements (e.g., devices, software platforms used), and design techniques to use is not just required capabilities but often the context in which the product is used. Analyzing and understanding different product usage contexts are very important for successful product line engineering.

What is important in the variability modeling is that:

- There are different market segments or user communities who may have different goals and/or different product usage context
- Different goals or usage contexts may require different quality attributes or capabilities
- Same capabilities may be implemented in different ways (design decisions), which may have different quality characteristics

In variability modeling, we explore these different dimensions and model relationships between modeling elements as shown in Fig. 2.1. In the following section, we review techniques for variability modeling.

4 Modeling Techniques

4.1 Feature Modeling

Since feature modeling [1] was first introduced two decades ago, it has been widely accepted by the software reuse and the software product line engineering (SPLE) communities as a means for modeling C&V of a product line, i.e., a family of products. This is because features are abstract concepts effectively supporting communication among diverse stakeholders of a product line, and therefore, it is

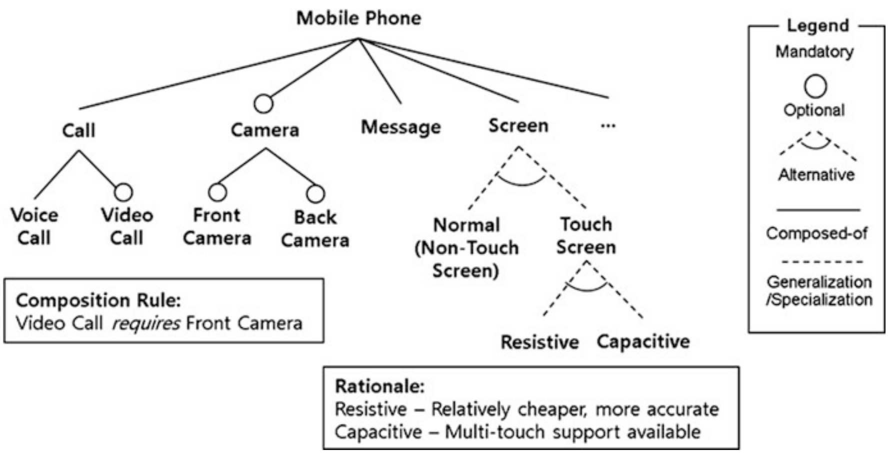

Fig. 2.2 A FODA feature model of a mobile phone product line

natural and intuitive for people to express commonality and variability (C&V) of product lines in terms of features. Also, it has been recognized that the C&V information codified by a feature model is most critical for developing reusable software assets.

In practice, many feature-based approaches to SPLE use features as units of:

- Capability that is delivered to customers
- Requirement containers, i.e., units of requirement specification
- Product configuration and configuration management
- Development and delivery to customers
- Parameterization for reusable assets, i.e., parameters for instantiating reusable assets
- Product management for different market segments

Furthermore, future products are typically discussed and described in terms of features gathered from market surveys, individual customers, research labs, or technology roadmaps.

The original feature model has very simple modeling primitives: structural relationships (composition, generalization/specialization), alternativeness, optionality, and mutual dependencies (inclusion, exclusion). Textual description and attributes of a feature may be defined. Also, the rationale for selection of an optional or alternative feature may be added as a textual description. Figure 2.2 shows an example of FODA feature model. This feature model describes a product line for mobile phones. In Fig. 2.2, *Video Call*, *Camera*, *Front Camera*, and *Back Camera* features are optionally selectable features. *Resistive* and *Capacitive* features are alternatives and can be thought of as specializations of general *Touch Screen* feature. As can be seen in the composition rule in Fig. 2.2, *Front Camera* feature must be selected when *Video Call* is selected. Selection of alternative features, *Resistive* and *Capacitive*, is made based on rationales shown in Fig. 2.2.

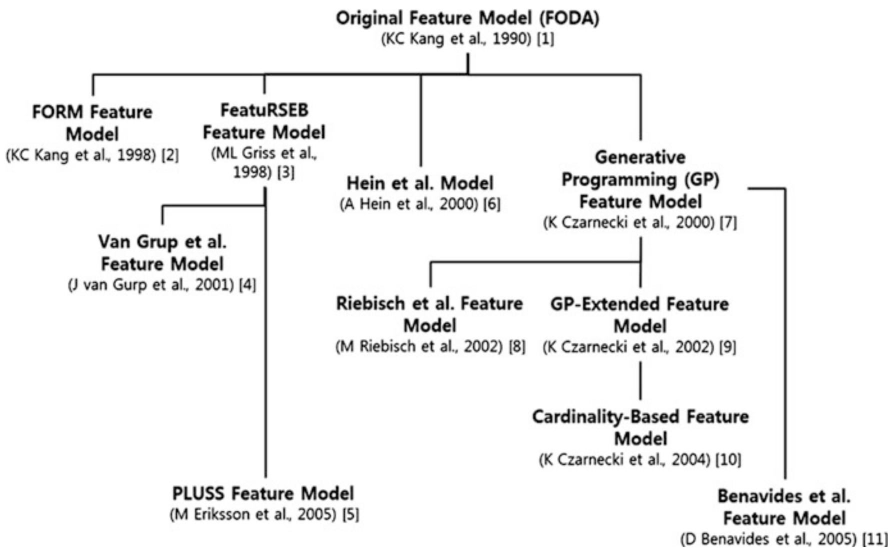
Original Feature Model (FODA)
(KC Kang et al., 1990) [1]
FORM Feature Model
(KC Kang et al., 1998) [2]
FeatuRSEB Feature Model
(ML Griss et al., 1998) [3]
Hein et al. Model
(A Hein et al., 2000) [6]
Generative Programming (GP) Feature Model
(K Czarnecki et al., 2000) [7]
Van Grup et al. Feature Model
(J van Gurp et al., 2001) [4]
Riebisch et al. Feature Model
(M Riebisch et al., 2002) [8]
GP-Extended Feature Model
(K Czarnecki et al., 2002) [9]
Cardinality-Based Feature Model
(K Czarnecki et al., 2004) [10]
PLUSS Feature Model
(M Eriksson et al., 2005) [5]
Benavides et al. Feature Model
(D Benavides et al., 2005) [11]

Fig. 2.3 Feature modeling approaches

For example, in case of buying a mobile phone, if a customer only concerns touch accuracy, then s/he may want to select *Resistive* rather than *Capacitive*.

After the introduction of FODA, many researchers have extended the feature model by adding new concepts for their researches [2–20], thereby resulting many variations (see Fig. 2.3) and extensions are still continuing. For instance, FODA was extended in [2] by introducing different viewpoints and grouping features into capability features modeling C&V of functions and services provided by the products, operating environment features modeling C&V of the environments in which these products are deployed and interface with, and domain technology and implementation techniques modeling important design decisions. A new relationship type "implemented by" was introduced to connect capability features (the functional dimension) with domain technology and implementation technique features (the design dimension) that may be used to implement capability features. Griss [3], Gurp [4], and Eriksson [5] made notational changes to the feature model and also provided notations for expressing dependencies and feature binding time. Hein [6] provided a UML-based modeling language. Czarnecki [7, 9, 10], Riebisch [8], and Benavides [11] refined the alternative relationship of FODA to XOR and OR relationships and also added the concept of cardinality allowing multiple selection of a feature. Attributes of features are also included in the feature model. Table 2.1 shows a summary of extensions made by each feature modeling approach.

As we examined the applications of these feature modeling approaches, we noticed that a feature model was often used to model different "concerns" of a product line in one model without delineating them. These concerns include the following: missions or business goals that need to be achieved by a product line,

Table 2.1 Summary of feature modeling approaches

Feature modeling approach	Extensions
FORM Feature Model [2]	• Introducing different viewpoints: capability, operating environment, domain technology, and implementation technology • Introducing a new relationship type *implemented by*
FeatuRSEB Feature Model [3]	• Making notational change: alternative features → variation point feature and variant features • Providing constraint (e.g., *require*) notation • Providing binding time notation: reuse-time and use-time binding
Van Gurp et al. Feature Model [4]	• Introducing *external* features • Refining the generalization/specialization relationship to OR-specialization and XOR-specialization relationships • Providing binding time notation: compile-time, link-time, and run-time binding
PLUSS Feature Model [5]	• Making notational changes: – A group of alternative features → single adaptors – A group of optional features → multiple adaptors • Providing constraint notation
Hein et al. Feature Model [6]	• Providing UML-based modeling language • Introducing secondary structure for constraint (e.g., *require*) dependencies
Generative Programming (GP) Feature Model [7]	• Refining the alternative relationship to XOR and OR relationships
Riebisch et al. Feature Model [8]	• Introducing the concept of feature group and group cardinality • Providing constraint notation
GP-Extended Feature Model [9]	• Introducing the concept of feature cardinality
Cardinality-Based Feature Model [10]	• Introducing the concept of feature cardinality, feature group, and group cardinality • Introducing a new relationship type *feature diagram reference*
Benavides et al. Feature Model [11]	• Including feature attributes in the feature model

functional capabilities provided by a product line, required nonfunctional properties (quality attributes), operating environments in which products are deployed, major design decisions to realize functional capabilities and achieve quality attributes, and rationales for configuring features for a certain usage context. These concerns may be classified as shown in Fig. 2.1.

This coexistence of multiple viewpoints[2] in a single model naturally leads to the following problems:

[2] For the same object, we can observe it from different angle, i.e., viewpoint, and extract different information. For example, an orthopedic doctor's view of human will be different from that of an internist.

- Analyzing, understanding, and defining the relationships between different viewpoints are a big burden to product line analyst
- Relationships between different viewpoints are not always explicitly defined
- A feature model with multiple concerns tends to become very complex, making it hard to comprehend and maintain
- The boundary between the problem space (features to capture the context of a product line) and the solution space (features to capture the services and design decisions of a product line) are not clearly distinguished
- Optimal configuration of products considering quality attributes is difficult

There is a need for a holistic approach [21] to feature modeling to alleviate these difficulties by first exploring the feature space to identify different concerns and divide it into subspaces based on different concerns and then to examine how they are related to each other, enabling product line analysts to examine a broad spectrum of concerns of a product line while focusing on specific concerns separately. By delineating these concerns as distinct viewpoints, analysis of a product line becomes thorough and systematic. This means that a product line analyst can concentrate on a specific modeling space with a clearly defined viewpoint (i.e., concern) at a time and then analyze and model relationships between different concerns later. An example of this holistic approach is shown in Sect. 5.

4.2 Decision Modeling

The decision modeling technique for modeling variability was introduced by [22].
A decision model consists of:

- Domain-related questions to be answered in developing products
- The set of possible answers/decisions to each question
- References to the affected artifacts and variation points, or references to the affected decisions
- Descriptions of the effect on the assets for each decision, or descriptions of the effects on the answer sets of the affected decisions

The decision modeling technique relates domain questions to other related domain questions and then, ultimately, to domain solutions which are variation points and/or variants. It focuses on capturing decisions to be made in configuring products. The feature modeling, however, focuses on exploring, understanding, and modeling the feature space (i.e., domain "questions"-problems and their solutions) of a domain in terms of commonalities, variabilities, and relationships among them. The rationale for each choice may be provided as textual description. Both modeling techniques may be used to configure products of a product line.

5 Variability Modeling: An Example

In this section, we further explore various dimensions of variability modeling explained in Sect. 3 using an Elevator Control System (ECS) product line as example [23, 24]. We will also see how these different dimensions are related to each other. The feature modeling technique is used in the exploration.

5.1 *Problem Space Exploration*

The problem space includes features for goals/objectives, usage contexts, and quality attributes of a product line as shown in Fig. 2.1. These features present the concrete context of a product line, i.e., external forces that drive selection of specific design decisions, i.e., architectures, algorithms, or implementation techniques; these problem features are important to understand real-world problems[3] that the product line should address. That is, the problem space captures the information of:

- Why is the product line required in the market?
- When is a certain product configuration used?
- What are the expected qualities of a specific product or the product line?

The answers to these questions should be captured in an exploitable form so that we can establish clear traceability, not starting from functional product features, but from real-world problems.

The problem space can be divided into three sub categories: goal/objective, usage context, and quality attribute features. The goal/objective features represent what a system should achieve in order to solve real-world problems. For example, in the ECS product line, the real-world problem is as follows: as multistory buildings are introduced and the number of floors increases, moving objects between floors becomes difficult. In order to solve this *real-world problem*, the goal/objective of ECS may be: "Move objects between different floors of a building in an *efficient* way." It is important to clearly define the goal as it implies the scope of the product line. The above goal, for instance, can also be achieved by an escalator. If it is not the intension and if we want to include only elevators, the goal should be refined as: "Move objects between different floors of a building *vertically* in an efficient way *using a cage with doors*" (see Fig. 2.4a). Through such refinement iterations, product line analysts, market analysts, and developers can establish an explicit boundary of a product line and can share a common understanding about the ultimate goal of the product line.

[3] In this chapter, we did not cover modeling real-world problems but focused on "external factors" derived from real-world problems that influence configuration of features in the solution space.

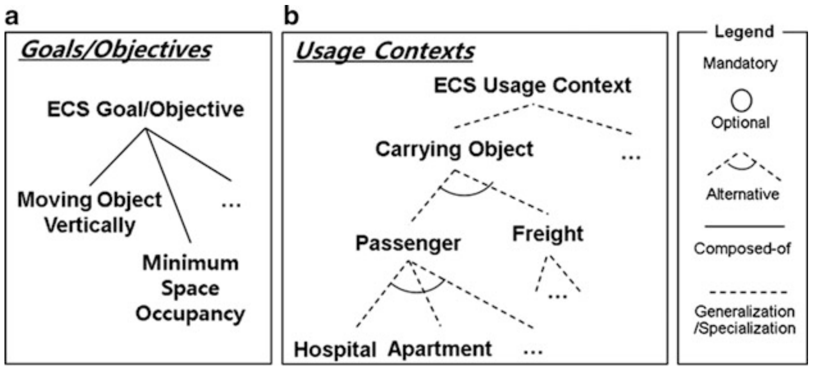

Fig. 2.4 (**a**) A goal/objective feature model and (**b**) a usage context feature model of the ECS product line

The next category is usage context, which represents a set of circumstances where a system is operated in. According to [25], usage contexts are any contextual setting in which a product is deployed and used. We follow this definition and it includes features about physical environments, user profiles, social or legal issues, business concerns, etc. For example, depending on the types of objects carried by an elevator, the usage context of ECS can be either a passenger elevator or a freight elevator (as shown in Fig. 2.4b).

The last category is about quality attributes: goal/objective and usage context features determine quality attribute features. Quality attribute features represent nonfunctional requirements that a system should satisfy while meeting its functional requirements. For example, for a passenger elevator, *Safety* and *Usability* features are important, while, for a freight elevator, "car call cancelation" feature may not be used for safety because of the weight of the load and the momentum of the elevator. Figure 2.5 shows an example of quality attribute feature model.

We need to explore C&V along these dimensions, which essentially derive decisions on required capabilities (functions) and various design choices.

In the following section, we discuss the solution space feature.

5.2 Solution Space Feature

The solution space captures functional, operational, and technical features that should be implemented for a product line. Most feature modeling approaches in the literature starts analyzing features that belong to this space, which can be classified into four categories (i.e., capability, operating environment, domain technology, implementation technique) according to FODA. It should be also noted that the term "solution" does not mean design artifacts in the space; features in this space are "solution decisions" for the problem space features, and these

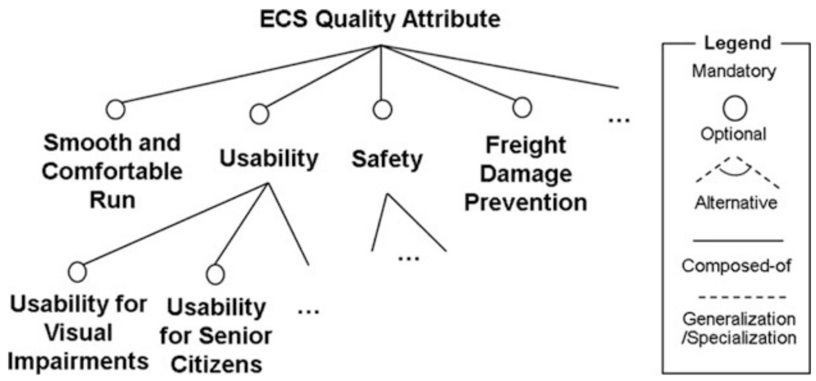
ECS Quality Attribute
Smooth and Comfortable Run
Usability
Safety
Freight Damage Prevention
...
Usability for Visual Impairments
Usability for Senior Citizens
...
...
Legend
Mandatory
Optional
Alternative
Composed-of
Generalization /Specialization

Fig. 2.5 A quality attribute feature model of the ECS product line

solution decisions must be implemented as product line assets (e.g., components) (The Artifact Space in Fig. 2.1). Figure 2.6 shows an example of solution space features.

Firstly, the capability features represent end-user visible characteristics of system such as service, operation, and function. For example, *Speed*, *Capacity*, *Hall Call Handling*, and *Motor Control* in Fig. 2.6a are capability features of the ECS product line. Secondly, the operating environment feature model captures C&V of target environments where products are deployed and operated in/on. For example, *RTLinux*, *VxWorks*, and *WindowsCE* in Fig. 2.6b are various real-time operating systems of the ECS product line. There are various sensors for detecting weight and leveling an elevator with building floors. Finally, design features represent design decisions such as domain technologies and implementation techniques. For example, in Fig. 2.6c, domain-specific algorithms such as *Motor Control Method* and *Weight Detection Method* are design decisions that are only meaningful in the ECS product line. Communication methods such as *TCP* and *UDP* represent concrete implementation techniques for a product line but they are more general and can be used in other product lines.

In the following section, we describe the relationships between these different viewpoints.

5.3 Dependencies Between Different Variability Viewpoints

In the variability modeling discussed in this section, features in the problem space *drive* decisions on features in the solution space. This means that the problem space features set clear contexts for identifying the solution space features and, thus, establishing explicit mapping between features in the two spaces. To model these spaces, we identified four activities and their relationships as depicted in Fig. 2.7.

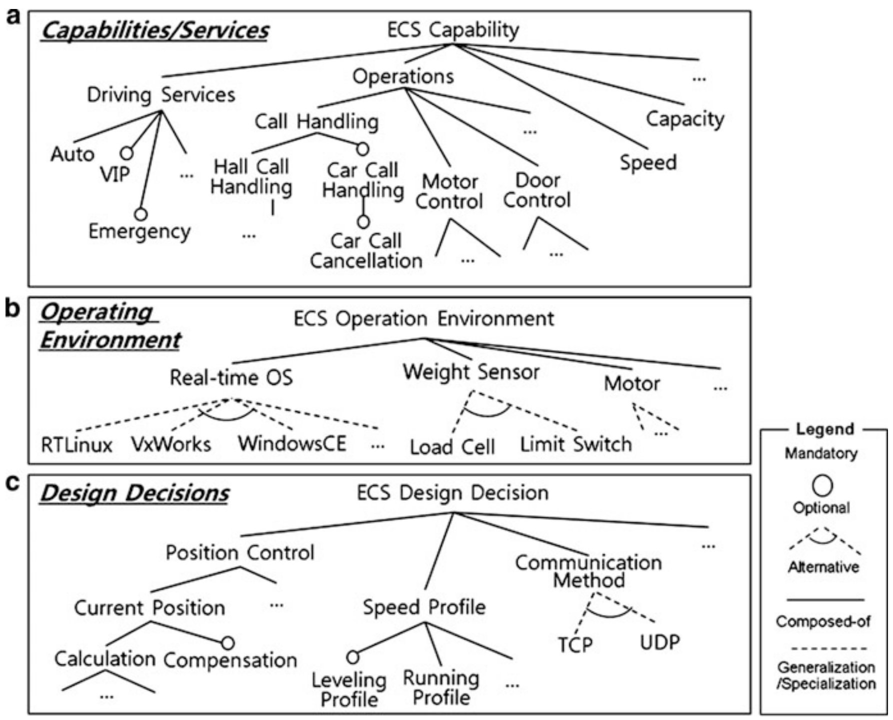

Fig. 2.6 A solution space feature model of the ECS product line

These activities are iterative and the arrows in Fig. 2.7 show data flow, i.e., use of work products at each activity. Each activity is briefly described below.

Organizing goal/objective features and usage context features from real-world problems of a product line initiates the modeling process. Goal/objective features specify the boundary of the product line and usage context features set specific contexts for the product line. The organized goal/objective features and usage context features are used as inputs to other activities.

In quality attribute feature modeling, quality requirements needed to achieve goals/objectives under various usage contexts are identified and organized into a quality attribute feature model. For example, the “safety” quality requirement of the ECS product line is to achieve the goal/objective of moving passengers safely in passenger elevators, and the “freight damage prevention” quality requirement is a goal set for freight elevators.

The problem space features (i.e., goal/objective, usage context, and quality attribute features) are used as primary inputs for the solution space feature modeling activity. Functional requirements that support the goal/objective under various usage contexts are identified as capability features. For example, the *Motor Control* capability feature is defined to satisfy the goal/objective of carrying objects between floors. The identified capability features may be refined further, and relevant domain technology and implementation features are identified considering

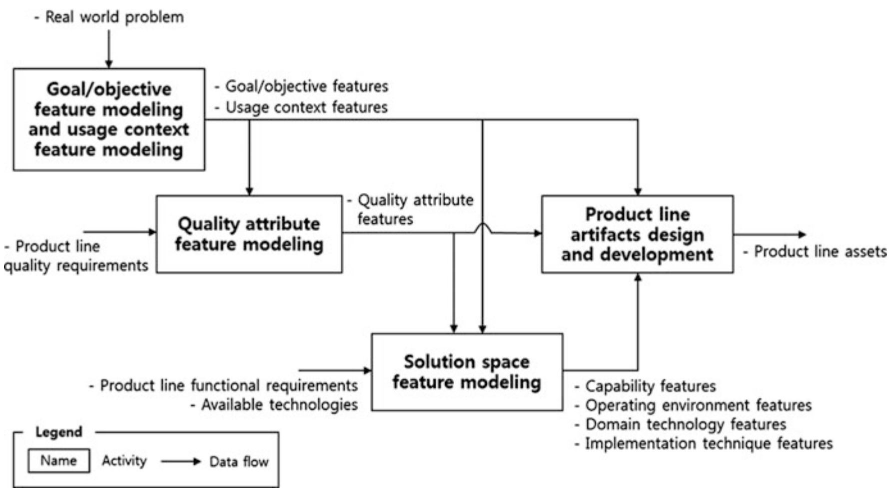
- Real world problem
Goal/objective feature modeling and usage context feature modeling
- Goal/objective features
- Usage context features
Quality attribute feature modeling
- Quality attribute features
- Product line quality requirements
Product line artifacts design and development
- Product line assets
Solution space feature modeling
- Product line functional requirements
- Available technologies
- Capability features
- Operating environment features
- Domain technology features
- Implementation technique features
Legend
Name
Activity
Data flow

Fig. 2.7 Variability modeling process

goals and quality features and modeled in a solution space feature model. For example, leveling profile techniques that support the "smooth and comfortable run" quality attribute are identified as domain technology features.

In the product line artifacts design and development activity, the identified solution space features are implemented as product line artifacts including product line architectures, objects, and code modules. Variabilities captured as optional/ alternative features in the solution space are embedded into the product line artifacts using various variability realization techniques (e.g., macro, aspects, etc.) [26–28].

In this section, we have examined the scope of variability. We will explore the temporal variability of product line software in the next section.

6 Feature Binding Time: Variability in Temporal Dimension

So far, we have seen C&Vs in the spatial dimension only, i.e., what features are common and what can vary. However, we should also explore C&Vs in the temporal dimension, i.e., when variability occurs, which is generally known as feature binding time. Generally, feature binding time has been looked at from the software development lifecycle viewpoint [7, 29], in which the focus has been given to the phase of the lifecycle at which a feature is incorporated into a product. In product line engineering, however, there exists another dimension that we have to consider, which we call *feature binding state* [12]. A feature may be *included* in the asset or a product at any product line lifecycle phase, but their *availability* for use can be determined at the time of inclusion or at any time after inclusion by enabling or disabling the included feature. Activation of the available features may

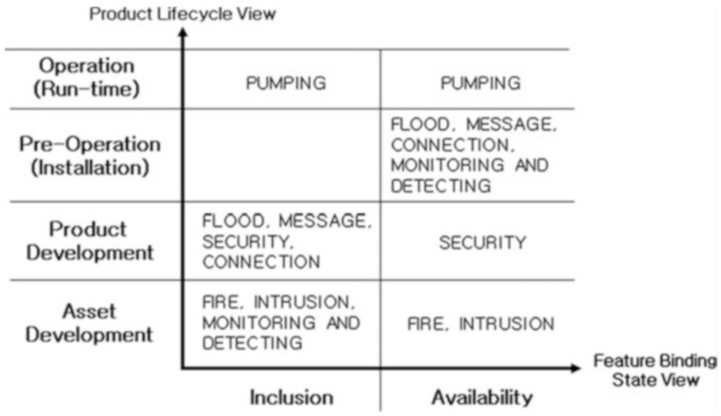
Product Lifecycle View
Operation (Run-time)
Pre-Operation (Installation)
Product Development
Asset Development
PUMPING
PUMPING
FLOOD, MESSAGE, CONNECTION, MONITORING AND DETECTING
FLOOD, MESSAGE, SECURITY, CONNECTION
SECURITY
FIRE, INTRUSION, MONITORING AND DETECTING
FIRE, INTRUSION
Feature Binding State View
Inclusion
Availability

Fig. 2.8 Feature binding time analysis

be controlled to avoid a feature interaction problem.[4] Thus, feature binding time analysis with an additional viewpoint on *feature binding state* (which includes *inclusion* and *availability* states) provides a more precise framework for feature binding analysis.

For the purpose of temporal variability analysis, we can simplify the product line lifecycle into four phases: asset development, product development, pre-operation, and operation (run-time), shown as the vertical axis in Fig. 2.8. The horizontal axis shows binding states. The example in Fig. 2.8 shows that both *FIRE* and *INTRUSION* features are included in assets, and they are available for use as soon as the assets are included in a product. However, *FLOOD* and *MESSAGE* features are included during the product development time as product-specific features, but their availability is determined at installation time. The *PUMPING* feature is included and becomes available at operation time (i.e., run-time binding).

7 Discussion

After the FODA method [1] was published, there have been various efforts to introduce different viewpoints for feature modeling based on their own experiences [2, 10, 12–20]. These extensions include structural, configuration, binding, operational dependency, and traceability viewpoints. For the structural viewpoint [2, 10, 13, 14, 17–19], extended feature specification, feature relationships, and feature categories [10, 13, 16–18] added strict or recommended constraints into a feature model for helping product feature configuration in the configuration viewpoint.

[4] The problem of unexpected side effects when a feature is added to a set of features is generally known as the feature interaction problem.

Lee and Kang [12] extended a feature model by introducing feature binding unit (i.e., groups of features bound together) with binding time and techniques. Fey et al. [14–16, 19, 20] identified various operational dependencies between features, such as activation dependency, modification dependency, etc. Kang et al. [2] defined implementation relationship (i.e., a feature is necessary to implement another feature) to model traceability between functional and design features. These extensions, however, are limited to solution space modeling. Kang et al. [21] extended the scope of feature modeling further to cover problem space modeling.

In FODA [1], it is stated that issues and decisions must be incorporated into a feature model in order to provide the rationales for choosing options and selecting among alternatives. However, how issues and decisions are modeled and how they are related to (solution space) features was not explained. Kang et al. [21] modeled issues and decisions as problem space features and explicitly captured the relationships between problem space features and solution space features. These relationships are used in product feature configuration.

In FOPLE [30], marketing and production plan (MPP) is introduced as rationales for identifying and selecting product features. MPP can include goal/objective features and usage context features (e.g., user profile and cultural/legal constraints of MPP are similar to usage context features). In FOPLE, it is stated that MPP provides quality attributes for architecture design and refinement. However, they do not discuss how MPP provides different quality attributes and how quality attributes affect selection of product features. In this chapter, we explicitly explain relationships among usage context features, quality attribute features, and product features.

Some researchers [31, 32] added a quality attribute viewpoint into feature model and associated quality attributes with solution space features. Yu et al. [31] proposed a goal model to capture stakeholder goals that may represent quality attributes and associate goals to features. Thurimella et al. [32] suggested issue-based variability model that combines rationale-based unified software engineering model [33], and orthogonal variability model [34]. In their model, quality attributes can be modeled as criteria for selecting product features. However, Yu et al. and Thurimella et al. did not discuss how product-specific quality attributes are identified. In [21], Kang et al. discussed how product-specific quality features are identified from product usage context features and product quality requirements.

Some researchers [25, 35] proposed usage context viewpoint into feature model and associate usage contexts with solution space features. Hartmann and Trew [35] introduced a context variability model and define dependencies (i.e., requires, excludes, and sets cardinality) between a context variability model and a feature model. Lee and Kang [25] proposed usage context variability model and quality attribute variability model and defined relationships among usage contexts, quality attributes, and product features; selection of variant usage contexts eliminates choices of variant quality attributes and those of variant product features, and selection of variant quality attributes eliminates choices of variant product features, which is similar to modeling discussed in this section. Kang et al. [21] adopted usage context analysis introduced in these papers [25, 35], but, unlike these papers, they clearly defined boundaries and relationships between the problem space, solution space, and artifact space.

Czarnecki et al. [10] suggested the concept of staged configuration, a process of specifying a family member in stages where each stage eliminates configuration choices, which can reduce the complexity of feature selection. Czarnecki et al. [36] extended this idea and introduce multi-level configuration, a form of staged configuration where the choices available to each stage are represented by separate feature models. In [36], it is stated that the criteria (e.g., geographical area or market segment) used to distinguish between the multiple product lines can be captured in a level-0 feature model, which is similar to usage context features discussed in this section. Their approaches [10, 36] are in the context of software supply chains [37] (i.e., each configuration stage is performed by different stakeholders in a software supply chain). Kang et al. [21] suggested a product feature configuration process that facilitates quality-based product configuration.

8 Summary and Outlook

This section introduces a holistic feature modeling method that enables product line analysts to capture complex concerns of a product line into different viewpoints and to decide product configuration systematically. Coexistence of multiple viewpoints in a single model without delineating them resulted in a highly complex and unmanageable feature model. The key idea in this section is the explicit separation of problem space features from solution space features. The approach also provides multiple viewpoints for each space so that a product line analyst can concentrate on a specific modeling space with clearly defined viewpoints at a time and do not need to consider other concerns. Relationships between these different viewpoints are explicitly modeled and used in making configuration decisions.

In this chapter, we explored explicit connections between goals/objectives, product usage contexts, quality attributes, and functional and design features. We also explored feature binding time issues. We expect to see more formal treatments of these subjects in a near future.

References

1. Kang, K.C., Cohen, S.G., Hess, J.A., Novak, W.E., Peterson, A.S.: Feature-oriented domain analysis (FODA) feasibility quality attributes and study. Technical report, CMU/SEI-90-TR-21, November 1990
2. Kang, K.C., Kim, S., Lee, J., Kim, K., Shin, E., Huh, M.: FORM: a feature-oriented reuse method with domain-specific reference architectures. Ann. Softw. Eng. **5**, 143–168 (1998)
3. Griss, M.L., Favaro, J., d'Alessandro, M.: Integrating feature modeling with the RSEB. In: 5th International Conference on Software Reuse, pp. 76–85 (1998)

4. van Gurp, J., Bosch, J., Svahnberg, M.: On the notion of variability in software product lines. In: Working IEEE/IFIP Conference on Software Architecture, pp. 45–54 (2001)
5. Eriksson, M., Börstler, J., Borg, K.: The PLUSS approach – domain modeling with features, use cases and use case realizations. In: Obbink, H., Pohl, K. (eds.) SPLC 2005. LNCS, vol. 3714, pp. 33–44. Springer, Heidelberg (2005)
6. Hein, A., Schlick, M., Vinga-Martins, R.: Applying feature models in industrial settings. In: 1st International Software Product Line Conference, pp. 47–70 (2000)
7. Czarnecki, K., Eisenecker, U.W.: Generative Programming: Methods, Tools, and Applications. Addison-Wesley, Reading, MA (2000)
8. Riebisch, M., Böllert, K., Streitferdt, D., Philippow, I.: Extending feature diagrams with UML multiplicities. In: 6th World Conference on Integrated Design & Process Technology, Pasadena, CA, USA, 23–27 June 2002
9. Czarnecki, K., Bednasch, T., Unger, P., Eisenecker, U.: Generative programming for embedded software: an industrial experience report. In: Batory, D., Consel, C., Taha, W. (eds.) GPCE 2002. LNCS, vol. 2487, pp. 156–172. Springer, Heidelberg (2002)
10. Czarnecki, K., Helsen, S., Eisenecker, U.: Staged configuration using feature models. In: Nord, R.L. (ed.) SPLC 2004. LNCS, vol. 3154, pp. 266–283. Springer, Heidelberg (2004)
11. Benavides, D., Trinidad, P., Ruiz-Cortés, A., Pastor, O., Falcao e Cunha, J.: Automated reasoning on feature models. In: CAiSE 2005. LNCS, vol. 3520, pp. 491–503. Springer, Heidelberg (2005)
12. Lee, J., Kang, K.C.: Feature binding analysis for product line component development. In: van der Linden, F. (ed.) PFE 2003. LNCS, vol. 3014, pp. 250–260. Springer, Heidelberg (2004)
13. Capilla, R., Dueñas, J.C.: Modeling variability with features in distributed architectures. In: van der Linden, F. (ed.) PFE-4 2001. LNCS, vol. 2290, pp. 319–329. Springer, Heidelberg (2002)
14. Fey, D., Fajta, R., Boros, A.: Feature modeling: a meta-model to enhance usability and usefulness. In: Chastek, G. (ed.) SPLC2 2002. LNCS, vol. 2379, pp. 198–216. Springer, Berlin (2002)
15. Lee, Y., Yang, C., Zhu, C., Zhao, W.: An approach to managing feature dependencies for product releasing in software product lines. In: Morisio, M. (ed.) ICSR 2006. LNCS, vol. 4039, pp. 127–141. Springer, Heidelberg (2006)
16. Zhang, W., Mei, H., Zhao, H.: A Feature-oriented approach to modeling requirements dependencies. In: 13th IEEE International Conference on Requirements Engineering, pp. 273–284 (2005)
17. Streitferdt, D., Riebisch, M., Philippow, I.: Details of formalized relations in feature models using OCL. In: 10th IEEE International Conference on Engineering of Computer-Based Systems, pp. 45–54 (2003)
18. Ye, H., Liu, H.: Approach to modelling feature variability and dependencies in software product lines. IEEE Proc. Softw. **152**(3), 101–109 (2005)
19. Ferber, S., Haag, J., Savolainen, J.: Feature interaction and dependencies: modeling features for reengineering a legacy product line. In: Chastek, G. (ed.) SPLC2 2002. LNCS, vol. 2379, pp. 235–256. Springer, Berlin (2002)
20. Lee, K., Kang, K.C.: Feature dependency analysis for product line component design. In: Bosch, J., Krueger, C. (eds.) ICSR 2004. LNCS, vol. 3107, pp. 69–85. Springer, Heidelberg (2004)
21. Kang, K., Lee, J., Lee, H.: A holistic approach to feature modeling. September 2011 (unpublished)
22. Atkinson, C., Bayer, J., Bunse, C., Kamsties, E., Laitenberger, O., Laqua, R., Muthig, D., Peach, B., Wust, J., Zettel, J.: Component-Based Product Line Engineering with UML. Addison-Wesley, London (2002)
23. Lee, K., Kang, K.C., Chae, W., Choi, B.W.: Feature-based approach to object-oriented engineering of applications for reuse. Softw. Pract. Exp. **30**(9), 1025–1046 (2000)

24. Kang, K.C., Donohoe, P., Koh, E., Lee, J., Lee, K.: Using a marketing and product plan as a key design driver for product line asset development. In: Chastek, G. (ed.) SPLC2 2002. LNCS, vol. 2379, pp. 366–382. Springer, Berlin (2002)
25. Lee, K., Kang, K.C.: Usage context as key driver for feature selection. In: Bosch, J., Lee, J. (eds.) SPLC 2010. LNCS, vol. 6287, pp. 32–46. Springer, Heidelberg (2010)
26. Svahnberg, M., Gurp, J., Bosch, J.: A taxonomy of variability realization techniques. Softw. Pract. Exp. **35**(8), 705–754 (2005)
27. Anastasopoulos, M., Gacek, C.: Implementing product line variabilities. In: Symposium on Software Reusability: Putting Software Reuse in Context, pp. 109–117 (2001)
28. Bachmann, F., Clements, P.C.: Variability in software product lines. Technical report, CMU/SEI-2005-TR-012, September 2005
29. Bosch, J., Florijn, G., Greefhorst, D., Kuusela, J., Obbink, J.H., Pohl, K.: Variability issues in software product lines. In: van der Linden, F. (ed.) PFE-4 2001. LNCS, vol. 2290, pp. 13–21. Springer, Heidelberg (2002)
30. Kang, K.C., Lee, J., Donohoe, P.: Feature-oriented product line engineering. IEEE Trans. Softw. Eng. **19**(4), 58–65 (2002)
31. Yu, Y., Lapouchnian, A., Leite, J.C.S.P., Mylopoulos, J.: Configuring features with stakeholder goals. In: 2008 ACM Symposium on Applied Computing, pp. 645–649 (2008)
32. Thurimella, A.K., Bruegge, B., Creighton, O.: Identifying and exploiting the similarities between rationale management and variability management. In: 12th International Software Product Line Conference, pp. 99–108 (2008)
33. Wolf, T.: Rationale-based unified software engineering model. Dissertation, Technische Universität München (2007)
34. Pohl, K., Böckle, G., van der Linder, F.: Software Product Line Engineering Foundations, Principles, and Techniques. Springer, Berlin (2005)
35. Hartmann, H., Trew, T.: Using feature diagrams with context variability to model multiple product lines for software supply chains. In: 12th International Product Line Conference, pp. 12–21 (2008)
36. Czarnecki, K., Helsen, S., Eisenecker, U.: Staged configuration through specialization and multi-level configuration of feature models. Softw. Process Improv. Pract. **10**(2), 143–169 (2005)
37. Greenfiled, J., Short, K.: Software Factories: Assembling Applications with Patterns, Models, Frameworks, and Tools. Wiley, Indianapolis, IN (2004)