Chapter 8
Variability in the Software Product Line Life cycle

Kyo C. Kang, Hyesun Lee, and Jaejoon Lee

What you will learn in this chapter
- *Understand the software product line life cycle*
- *Understand the issues in variability management*

1 Introduction

Product line (PL) engineering is a 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 [1–3]. Two major activities of PL software engineering are core asset development (i.e., PL engineering) and product development (i.e., application engineering) using the core assets.

For the core asset development, PL requirements are essential inputs. These inputs, though critical, are not sufficient for PL asset development; a Marketing and Production Plan (MPP), which includes guidelines/plans on what features are to be packaged in products, how these features will be delivered to customers, and how the products will evolve in the future. Therefore, it is essential to include a marketing perspective into the PL variability analysis and explores requirements and design issues from the marketing perspective [4, 5]. With an MPP, reuse is not opportunistic; it is carefully planned for specific product and market(s).

Once commonalities and variabilities (C&V) of market needs and their requirements are analyzed and modeled, this information is used to develop software assets, i.e., architectures and components, with appropriate variation points

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

J. Lee
School of Computing and Communications, Lancaster University, Lancaster, UK
e-mail: j.lee3@lancaster.ac.uk

and variants. Once software assets have been developed for a PL, the product development phase involves adaptation and instantiation of these assets for a product. (Asset management is an ongoing process, which includes C&V analysis, and reengineering and refactoring of software assets.)

One of the most difficult and critical tasks in product line engineering is variability management. Various product line lifecycle products, including models, architectures, and components, have C&Vs, which are related vertically among elements within each lifecycle product and horizontally across different lifecycle products. C&Vs must be explored and modeled thoroughly and their consistencies must be maintained while PL evolves. As lifecycle products and their C&Vs are related vertically and horizontally, it is very difficult and also costly to manage and maintain their consistency.

Variability management from the perspective of MPP is the key aspect for managing variabilities of assets because assets are instantiated, adapted, and integrated to support MPP based on market needs. "Features" are abstractions of capabilities or functions that the customer wants/needs. Therefore, feature is the unit of delivery and also the unit of configuration and variability management

In this chapter, we explore various issues of C&V across the entire PL life cycle. As a starting point, Sect. 2 describes PL engineering activities and their inputs/outputs. The elements of MPP are explained and illustrated in Sect. 3 using an elevator controller example described in the box below, and Sect. 4 includes a discussion on how product line "problems" are modeled. The solution space modeling is included in Sect. 5, followed in Sect. 6 by a discussion of how product line artifacts such as architecture and components are designed based on the solution space models. Sections 7 and 8 include a discussion and conclude this chapter, respectively. It should be noted that the Feature-Oriented Reuse Method (FORM) has been used throughout the chapter for the purpose of illustration of various issues in product line variability management.

2 PL Lifecycle Activities: An Overview

PL engineering consists of two major engineering processes: PL asset development and product/application development. (See Fig. 8.1 for activities and their relationships.) The PL asset development consists of activities for analyzing PL problems (analyzing market needs, developing a marketing and product plan (MPP)); modeling C&V of the PL problems (goals aimed to be achieved by the products, product usage contexts, required quality attributes, etc.); solution modeling which includes exploration and modeling capabilities required by PL products and modeling PL requirements, exploring and modeling important design decisions that have significant quality implications, including domain technologies, COTS, external devices, etc.; and developing PL asset architectures and components based on the analysis results. The product development phase consists of a number of activities for analyzing product goals and usage contexts, analyzing product requirements and configuring the product (i.e., variability instantiation,

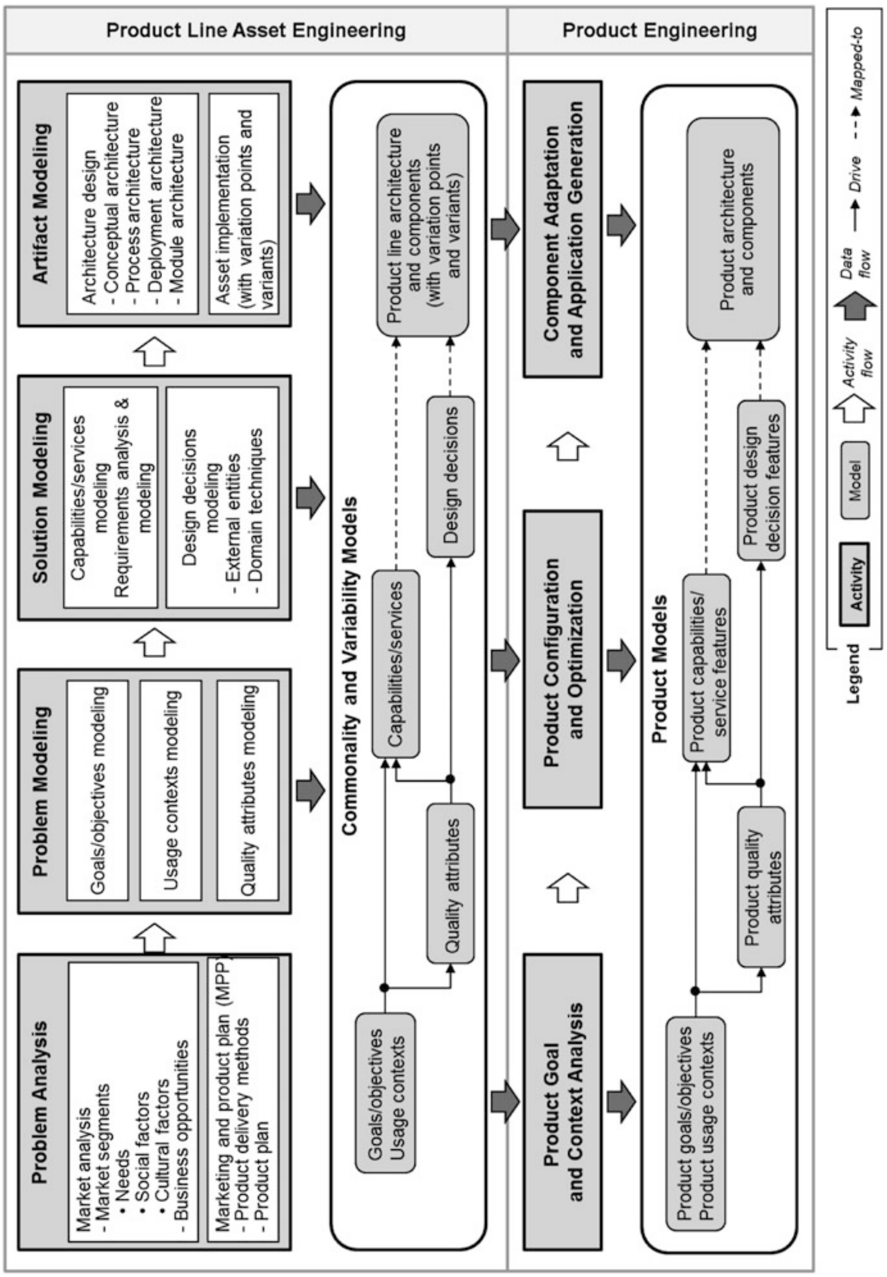

Product Line Asset Engineering
Product Engineering
Problem Analysis
Market analysis
- Market segments
• Needs
• Social factors
• Cultural factors
- Business opportunities
Marketing and product plan (MPP)
- Product delivery methods
- Product plan
Problem Modeling
Goals/objectives modeling
Usage contexts modeling
Quality attributes modeling
Solution Modeling
Capabilities/services modeling
Requirements analysis & modeling
Design decisions modeling
- External entities
- Domain techniques
Artifact Modeling
Architecture design
- Conceptual architecture
- Process architecture
- Deployment architecture
- Module architecture
Asset implementation (with variation points and variants)
Commonality and Variability Models
Goals/objectives
Usage contexts
Quality attributes
Capabilities/services
Design decisions
Product line architecture and components (with variation points and variants)
Product Goal and Context Analysis
Product Configuration and Optimization
Component Adaptation and Application Generation
Product Models
Product goals/objectives
Product usage contexts
Product quality attributes
Product capabilities/ service features
Product design decision features
Product architecture and components
Legend
Activity
Model
Activity flow
Data flow
Drive
Mapped-to

Fig. 8.1 PL engineering process and variability analyses

i.e., feature selection), selecting an appropriate architecture and components, and making necessary adaptations to the selected architecture and components, and generating/developing the product.

Elevator Example.

Elevator control systems are very familiar to everyone and it is easy to understand their behavior from an end user's perspective. However, the internal control and complex interfaces with external devices make it hard to build reliable and reusable software. Elevator products can largely be classified into passenger elevator or freight elevator depending on their main purpose of use. Since passenger elevators have a goal of carrying passengers comfortably, services or devices for comfortable ride are required. Of passenger elevators, those used in hospitals require more stringent floor leveling requirements and a low speed profile for patients on a wheelchair, while elevators in office buildings are less stringent on floor leveling but require a high speed profile. On the other hand, freight elevators do not necessarily require comfortable ride. They instead focus on carrying heavy loads safely, which requires special operational functions and devices for handling heavy loads. As can be seen, there is a wide range of quality requirements, which requires us to select various techniques, devices, and algorithms.

Analyzing the targeted markets and developing an MPP is the starting point for launching a PL. The diversity of market needs drives the development of a family of products, which is reflected in a marketing plan and product plan (MPP). Therefore, MPP serves as a key driver in PL asset development and variability management.

In C&V modeling, product features from MPP, which include functional (e.g., capabilities, services) and nonfunctional (e.g., product goals, product usage contexts, quality attributes) features, are organized into an initial C&V model (We use feature model for illustration. Other modeling techniques such as decision modeling may also be used.) which is refined through product line requirements analysis and then extended further with design features (i.e., operating environment, domain technology, and implementation technique features) as we follow the asset engineering activities. During PL requirements analysis [6], we elicit and organize PL requirements in terms of a PL use case model (variability expressed in terms of <extends> and <includes> stereotypes) and a PL object model, which are used in architecture modeling and component development.

The conceptual architecture design activity allocates features to architectural components and specifies data and control dependencies between architectural components. The result is a "conceptual architecture.[1] " A design object model

[1] The conceptual architecture describes a system in terms of abstract, high-level components and relationships between them.

must be developed based on the conceptual architecture, feature model, PL requirements, and other information such as commercial off-the-shelf (COTS) components and design patterns [7] that are relevant to the PL.

The conceptual architecture is refined into "process and deployment architectures[2]" by allocating components to concurrent processes and network nodes, considering whether to replicate each process or not, and defining methods of interactions between processes. Then the component design activity refines the process and deployment architectures into concrete components by using the design object model.

The MPP provides quality attributes for the architecture design and refinement. For example, the user profile information (i.e., the skill levels and computing environments of potential users) in the MPP is useful in determining the quality attributes (i.e., usability, scalability, etc.) required for the architecture design of the products targeted for each market segment. Also, the MPP is used in exploring design alternatives in the component design for feature delivery method support, feature interaction problem[3] resolution, etc.

The PL engineering processes are iterative and incremental, and repeat until we come to a design that has enough details for implementation. (The arrows in Fig. 8.1 show data flows, i.e., work products generated and used by each activity.) Details of each PL asset development activity are illustrated in the following section.

3 Problem Analysis

Problem analysis starts by exploring the market and developing a marketing and product plan for a product line. This activity initiates PL asset development; an MPP sets a specific context for PL analysis and reuse exploration in the PL. Products developed without consideration of user's needs and their capabilities and how they will be marketed will not be "sold." Functionality alone does not sell. Products must be configurable to meet the needs of and services required by users. Variation points embedded into the PL assets and mechanisms used to implement variants must be able to support the marketing and product plan effectively and efficiently.

3.1 Elements of MPP

The first part of an MPP is a marketing plan, which includes a market analysis with a C&V assessment of a market, and a marketing strategy for realizing business opportunities in the market (see the left portion of Fig. 8.2). The market analysis

[2] The process architecture represents a concurrency structure in terms of concurrent processes (or tasks) to which functional elements are allocated; the deployment architecture shows an allocation of processes to hardware resources.

[3] When products are developed with integration of components implementing various features, these features may interact with each other. The problem of unexpected side effects when a feature is added to a set of features is generally known as the feature interaction problem.

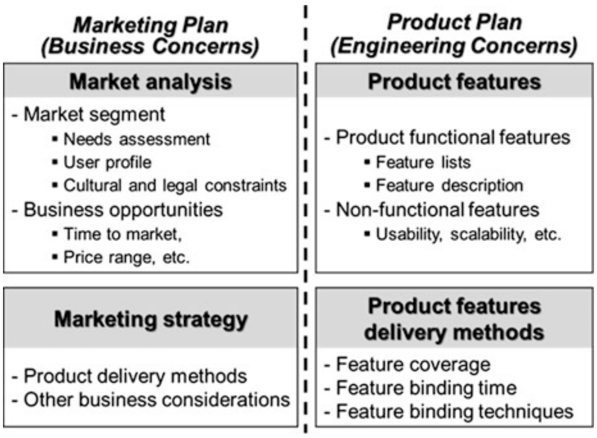

Marketing Plan (Business Concerns)
Product Plan (Engineering Concerns)
Market analysis
- Market segment
▪ Needs assessment
▪ User profile
▪ Cultural and legal constraints
- Business opportunities
▪ Time to market,
▪ Price range, etc.
Product features
- Product functional features
▪ Feature lists
▪ Feature description
- Non-functional features
▪ Usability, scalability, etc.
Marketing strategy
- Product delivery methods
- Other business considerations
Product features delivery methods
- Feature coverage
- Feature binding time
- Feature binding techniques

Fig. 8.2 Elements of a marketing and product plan

includes, for each market segment, a C&V assessment of needs, an analysis of potential users, cultural and legal constraints, the time to market, and the price range. Identifying market segments and needs of each market segment may be an iterative process. For example, we may start with the entire elevator systems market but then quickly realize that there are features specific only to elevators in office buildings or to freight elevators. We then focus on analyzing features peculiar to each market segment.

The marketing strategy initially includes an outline of product delivery methods with the information on how products will be delivered to customers and other business considerations. For example, customers may start with products with only core features but then add other features incrementally. Incremental features may be added to the products over the network. If this is the marketing strategy, the product delivery method and variability management method must be able to support this strategy.

Once the marketing plan part of the MPP has been defined, it is important to spend some effort to identify the characteristics of products in a PL in terms of features and develop a plan for incorporating the features. A product plan includes product features and product feature delivery methods (see the right portion of Fig. 8.2).

Product features are distinctive characteristics of products of a PL. They are largely classified into functional and nonfunctional features. Functional features include services, which are often considered marketable units or units of increment in a PL, or operations, which are internal functions of products that are needed to provide services. For example, automatic control, manual control, and VIP service features in elevator PL are functional features. Nonfunctional features include end-user visible application characteristics that cannot be identified in terms of services or operations, but as presentation, capacity, quality attribute, usage, cost, etc. For example, safety, reliability, and fault tolerance are important quality attributes for the elevator PL.

A product feature delivery method defines how product features will be "sold" or delivered to the customers and users, and how they will be installed and maintained. The product delivery method must support the marketing strategy. Some features may be prepackaged in all products as "standard features," and others may be selected at the product negotiation time. There may yet be other features that are specific to a customer and are built into the custom-made product.

3.2 Marketing and Product Planning: A ECS Product Line Example

Market segments affect engineering of assets for the ECS product line. For the purpose of illustration, we assume that the company's marketing strategy is to target the high-rise office building market with high-end products, the general hospital market with mid-level products, and the apartment market with low-end products, based on an analysis of market competitiveness.

Table 8.1 provides/is an example of MPP for the ECS PL. (The actual MPP should be a document describing all elements identified in Sect. 3.1; a simplified example is shown in this paper for illustration of the concept.) The user/maintainer profiles for each market segment are as follows:

- High-end market segment of high-rise office building uses: Dedicated engineers with computer science background are available for maintenance. The computing environment is distributed over the network and maintainers can access the system remotely.
- Mid-level market segment of general hospital uses: No computer skill is assumed for maintainer and ECS software should run on PCs they already have.
- Low-end market segment of household uses: No computer skill is assumed for maintainer and ECS software should run on PCs they already have.

The laws and the cultural traits of each country must also be identified in the marketing plan. For example, standards related to earthquake resilience, fire standards, electrical wiring rule, etc., may vary from country to country. Also, the safety and reliability requirements (e.g., the doors must remain open in case of a fire event) may be different.

Since the high-rise office building ECS has many customer-specific requirements, the "feature selection" method is chosen to adapt and integrate the requirements at the product delivery time. For the general hospital ECS and the apartment ECS, "prepackaged" method with a user-friendly interface is adopted for the users who do not have any computer knowledge.

The product delivery methods are refined to product feature delivery methods, which can be looked at from what features are allowed (feature coverage), when they are incorporated into the product (feature binding time: product build time, product delivery/installation time, or runtime), and how the feature incorporation is made (feature binding techniques: framework, template, load-table, plug-ins, etc.) [8–10]. For example, the apartment ECS have a closed set of features and the feature

Table 8.1 A marketing and product plan example for an ECS PL

Marketing and product plan for ECS product line			
Market segments	High-rise office building	General hospital	Apartment
User/ maintainer profile	Dedicated engineers with computer science backgrounds	No computer knowledge is assumed	No computer knowledge is assumed
Product delivery method	Feature selection from a predefined set of features (feature selection method)	Prepackaged method	Prepackaged method
Legal constraints	Because elevator is part of a building, it must comply with standards relating to earthquake resilience, fire standards, electrical wiring rule, etc.		
Product features	Call handling, indication handling, door control, motor control, hall call cancellation, car call cancellation, emergency driving, group management, etc.	Call handling, indication handling, door control, motor control, hall call cancellation, car call cancellation, emergency driving, hospital emergency, etc.	Call handling, indication handling, door control, motor control, etc.
Quality attributes	Door safety, usability, efficiency, emergency safety	Door safety, usability, smooth and comfortable run, position accuracy, emergency safety	Door safety, usability
Product feature binding time	Product delivery time	Product build time	Product build time

binding occurs at product build time in order to support the prepackaged product delivery method. For the high-rise office building ECS, however, customers may select any features from a predefined list, and feature binding occurs at product installation time using a load table that contains parameter values for instantiation.

4 Problem Modeling

Features in the problem model represent the concrete context of products of a product line, i.e., external forces that drive selection of specific architectures, implementation algorithms, or implementation techniques; these features are important to understand real world problems 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 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 product functional features, but from the context of a product line.

The problem space can be divided into three disjoint viewpoints: goal/objective, usage context, and quality attribute.

4.1 Goals/Objectives and Usage Contexts Modeling

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 and scope of the product line and usage context features set specific contexts for the product line. Figure 8.3 shows an example of them.

Our experience shows that we could explore the following areas while analyzing goals/objectives of a product line; (1) real world problems that the product line addresses, (2) other product lines that address the problems in a similar but different ways, (3) potential benefits that can be accrued from other product lines, and (4) key nonfunctional properties of the product line (e.g., comfortability, efficiency, etc.) that should be achieved. This information is used as an important input for the usage context feature modeling.

Next we analyze the usage context of a product line. The usage context features are to capture various usage contexts of products of the product line. For example, we identified usage context features *Passenger* and *Freight* and each usage context defines the objects to be carried by an elevator. (See *Carrying Object* in Fig. 8.3.) If we select *Passenger*, the scope of the ECS product line is restricted to *Passenger-ECS* products and irrelevant features are removed from selectable ones. If we look further into the product usage, we may be able to identify more specific product usages where different quality attributes and/or functions are needed. For example, an elevator in a hospital may carry patients on wheelchairs, and floor leveling and emergency button are important features. (See *Hospital* in Fig. 8.3 and quality features in Table 8.2.)

With the identified goal/objective and usage context features, the next activity is to analyze quality attribute features.

4.2 Quality Attributes Modeling

In this activity, we analyze quality requirements of a product line and model them as quality attribute features. We can use software requirements specification (SRS), quality requirements document, etc., as inputs of this activity. Suppose, for

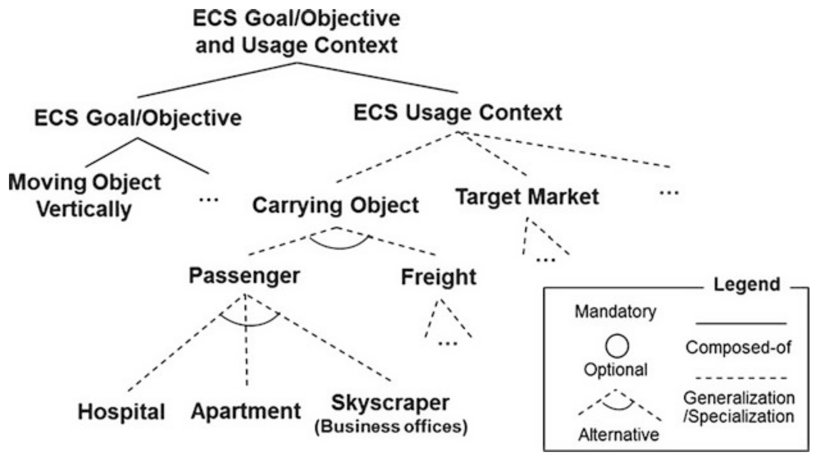


Fig. 8.3 A goal/objective and usage context feature model of the ECS PL

Table 8.2 Relationship between goal/objective, usage context, and quality attribute features of the ECS PL

Quality attribute features Goal/objective and usage context features	Door safety	Usability	Smooth and comfortable run	Position accuracy
{Passenger}	V	V	O	O
{Passenger, hospital}	V	V	V	V
{Passenger, skyscraper}	V	V	O	O

V (required), O (selectable)

example, that one of the quality requirements of ECS is: "If the doors of a cage detect a certain level of friction when the doors are closing, the ECS should open them immediately." From this statement, we can identify the quality attribute feature *Door Safety*. Figure 8.4 shows an example of quality attribute feature model.

Goal/objective and usage context features that we identified in the previous activity have specific quality implications. If a set of goal/objective and usage context features are selected for a product configuration, these features require a certain set of quality attribute features for the product configuration (i.e., the set of selected goal/objective and usage context features *drive* the quality attribute features). We can represent this relation explicitly using a table. For example, in Table 8.2, when we select the usage context feature *Passenger*, the quality attribute features *Door Safety* and *Usability* should be included in a product configuration; this is because user safety and convenience should be guaranteed for passenger ECS products. (See the second row in Table 8.2.) If the usage context feature *Hospital* (which is a specialization of *Passenger*) is selected, the additional quality attribute features *Smooth and Comfortable Run* and *Position Accuracy* should be included in

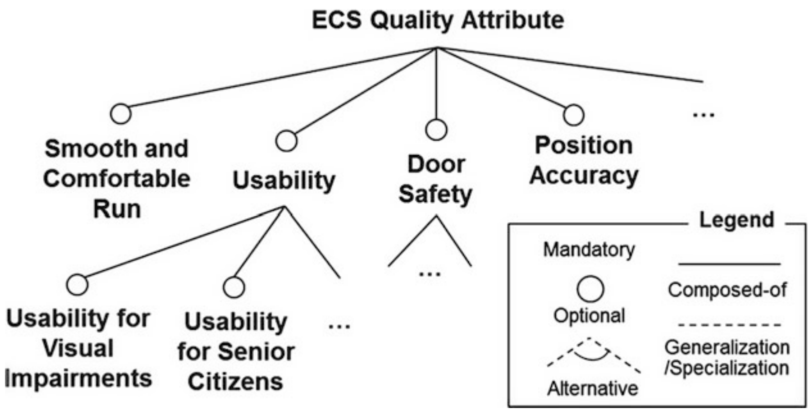

ECS Quality Attribute
Smooth and Comfortable Run
Usability
Door Safety
Position Accuracy
...
Usability for Visual Impairments
Usability for Senior Citizens
...
...
Legend
Mandatory
Optional
Alternative
Composed-of
Generalization /Specialization

Fig. 8.4 A quality attribute feature model of the ECS PL

a product configuration for moving patients on wheelchairs or beds safely. (See the third row in Table 8.2.)

The goal/objective, usage context, and quality attribute features capture the problem space of a product line. In the next activity, we analyze the solution space.

5 Solution Modeling

5.1 *Capabilities/Services and Design Decisions Modeling*

We analyze the solution space to satisfy requirements captured by the problem space features. The solution space includes features of the capability/service (e.g., *driving services*, *operations*, etc.) and design decision viewpoints (e.g., *position control*, *speed profile*, etc.). Figure 8.5 shows an example of solution space features.

After we finish feature modeling for the problem and solution spaces, we establish mappings between the problem and solution space feature models. Details are in [11].

5.2 *Relationships Between Problem Space Features and Solution Space Features*

Goal/objective and usage context features may require specific capability features. We can capture these relationships using a table. In Table 8.3, for example, the usage context feature *Hospital* requires context-specific functional requirements such as “when a patient triggers an emergency alarm, the ECS should call nurses/ doctors and stop the elevator in the nearest floor.” Therefore, when we select the

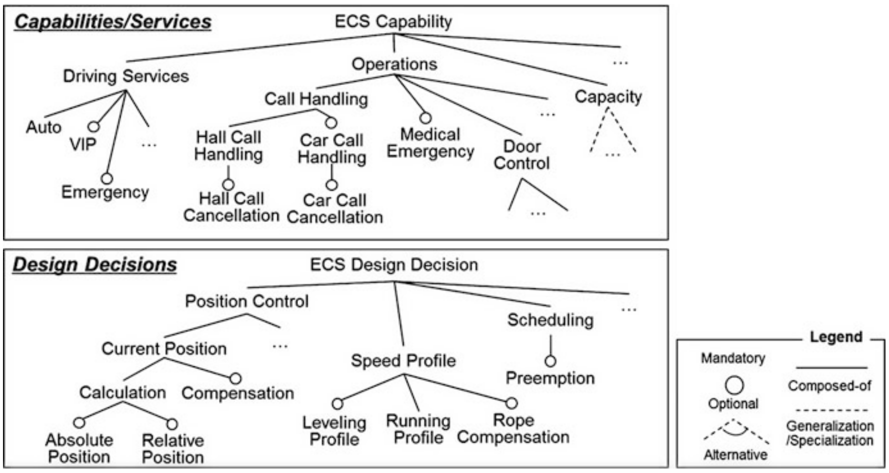


Fig. 8.5 A solution feature model of the ECS PL

Table 8.3 Relationship between goal/objective, usage context, and capability/service features of the ECS PL

Capability/service features Goal/objective and usage context features	Medical emergency	Double-deck	Door hold	Low speed (about 60 m/min)	Large capacity for car (about 3,000–5,000 kg)
{Passenger}	O	O	O	O	X
{Passenger, hospital}	V	O	V	O	X
{Passenger, skyscraper}	O	V	O	X	X

V (required), X (excluded), O (selectable)

usage context feature *Hospital*, the required capability features (e.g., *medical emergency*) must be included in a product configuration and irrelevant features (e.g., *large capacity for car*) must be removed from selectable features. (See the third row in Table 8.3.)

For each quality attribute feature, we should also establish a mapping to the solution space. A capability or a design feature may work for or against each quality attribute feature, and we capture the "strength" of this relation using qualitative measures such as *strongly support* (++), *weakly support* (+), *hurt* (−), and *break* (− −) [12]. This relation is represented using a table as shown in Table 8.4: the table shows relationships between quality attribute features and solution space features in the ECS product line. The design decision feature *Absolute Position* strongly (i.e., ++) *supports* the quality attribute feature *Position Accuracy* and *hurts* (i.e., −) the quality attribute feature *Low Cost*, whereas the design decision feature *Relative Position* weakly (i.e., +) *supports* both *Position Accuracy* and *Low Cost*. If the quality attribute feature *Position Accuracy* has the highest priority for an ECS product, both of the

Table 8.4 Relationship between quality attribute and solution space features of the ECS PL

Quality attribute features Solution space features		Position accuracy	Low cost
Calculation	Absolute position (optional)	++	–
	Relative position (optional)	+	+
Compensation of current position		+	–

++ (strongly support), + (weakly support), – (hurt), – – (break)

design decision features *Absolute Position* and *Relative Position* may be selected, although *Absolute Position hurts Low Cost*. However, if the quality attribute features *Position Accuracy* and *Low Cost* have the same priority, then only *Relative Position* can be selected.

5.3 PL Requirements Analysis

During the PL requirements analysis, functionalities that PL products must provide are captured using a set of models such as a use case model, an object model, etc. [6] A use case model defines interactions between the user and the system; an object model defines allocation of responsibilities. Other models may also be included depending on the domain that a PL is in. Based on this information, a PL component design provides a realization of common functions that can be used across the PL products.

Figure 8.6 shows a use case model of the ECS PL. Each of use case may embed optional/alternative features in the solution space. When we establish the mapping relation, we propose two different types of variability embedded in domain objects, and they are external and internal variability types. In Fig. 8.6, a UML stereotype-based notation introduced by Lee et al. [13] is used for expressing these two types. Other UML-based variability mechanisms such as [14] may also be used.

The external-variability type denotes that an associated use case is entirely related to the specified feature and inclusion/exclusion of the use case depends on the selection result of the feature. For example, '<<●*Car Call Handling*>>' in the *Process Car Calls* use case in Fig. 8.6 indicates that the inclusion of the *Process Car Call* use case depends on the selection result of the optional feature *Car Call Handling*. In other words, if *Car Call Handling* is selected for a product configuration, the *Process Car Call* use case should be included in a product; otherwise it should be removed from the product configuration.

The internal variability type means that a corresponding feature is partially related to the associated use case and specifics on how the feature is related to the use case can be found by looking inside of the use case. For example, "<<○*Hall Call Cancellation*>>" in the *Process Hall Calls* use case in Fig. 8.6 means that *Process Hall Calls* is related to the optional feature *Hall Call Cancellation*, and if we select or not select *Hall Call Cancellation*, the internal interaction(s) of the *Process Hall Calls* use case changes.

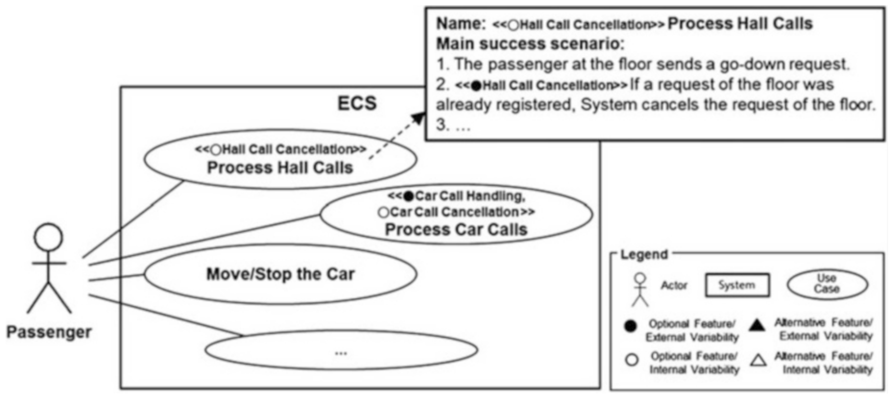

Name: <<○Hall Call Cancellation>> Process Hall Calls
Main success scenario:
1. The passenger at the floor sends a go-down request.
2. <<●Hall Call Cancellation>> If a request of the floor was already registered, System cancels the request of the floor.
3. …
ECS
<<○Hall Call Cancellation>> Process Hall Calls
<<●Car Call Handling, ○Car Call Cancellation>> Process Car Calls
Move/Stop the Car
…
Passenger
Legend
Actor
System
Use Case
Optional Feature/ External Variability
Alternative Feature/ External Variability
Optional Feature/ Internal Variability
Alternative Feature/ Internal Variability

Fig. 8.6 A product line use case model of the ECS PL

In object modeling, we first identify domain objects, which are candidate-reusable objects derived from the feature models based on the guidelines proposed in [11]. Then variabilities captured as optional/alternative features in the solution space are embedded into the object model.

Figure 8.7 shows an example of mappings between solution space features and domain objects. We also use the external and internal variability types. For example, "<<●*Car Call Handling*>>" in the *Car Call Handler* domain object in Fig. 8.7 indicates that if *Car Call Handling* is selected for a product configuration, the *Car Call Handler* object should be included in a product; otherwise it should be removed from the product configuration. For another example, '<<○*Car Call Cancellation*>>' in the *Car Call Handler* domain object in Fig. 8.7 means that if we select or unselect *Car Call Cancellation*, the internal implementation of the *Car Call Handler* object changes.

Once the feature model is refined and PL requirement models are developed, this information is used to refine the MPP as described in Fig. 8.1. Since the initial MPP only contains delivery methods for functional and nonfunctional features, product feature delivery methods for design features (e.g., operating environment features) should be developed during the refinement.

6 Artifact Modeling

After we establish the problem and solution space feature models and their relations, the next activity is to develop product line artifacts based on the feature models. During this activity, the identified solution space features are implemented as product line artifacts including product line architectures and asset implementation.

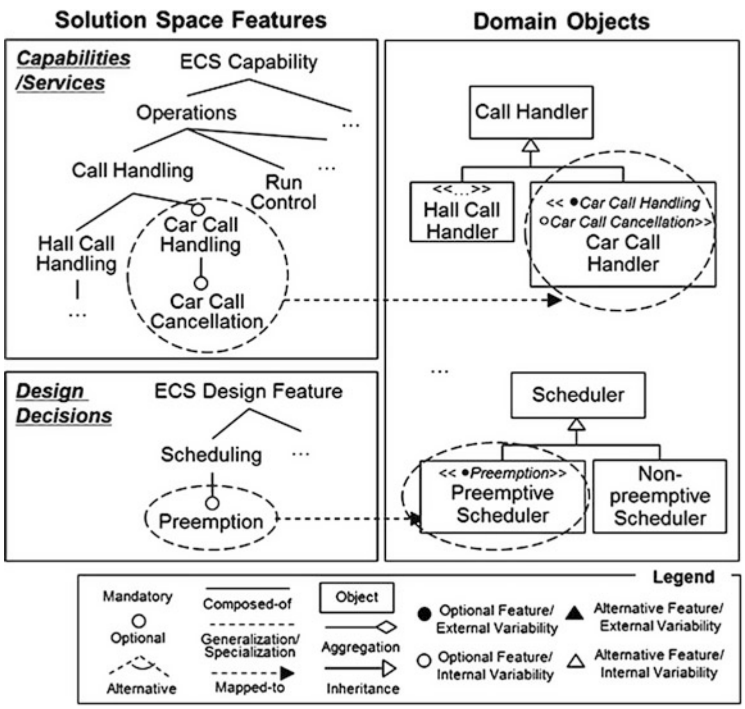


Fig. 8.7 Solution space features and domain objects mappings of the ECS PL

6.1 Architecture Design

In FORM [15], architecture design starts with identifying high-level conceptual components and specifying data and control dependencies among them. During the architecture design activity, the MPP is used as a key design driver. For example, the conceptual architecture for the high-rise office building ECS (see the conceptual architecture in Fig. 8.8.) consists of three major components (i.e., *Call Handler*, *Cage Operation Manager*, and *Safety Manager*) and the *Safety Manager* component is added to meet the quality requirement *Emergency Safety*. The *Safety Manager* monitors various safety related sensors (e.g., smoke sensor) and when an emergency situation (e.g., fire, earthquake) is detected, it notifies users via *Safety Supervision System* and sends the emergency status to the *Call Handler* and the *Cage Operation Manager*. The *Call Handler* receives call requests and schedules floors to visit and the *Cage Operation Manager* controls various external objects (e.g., door, motor) in elevator cars. When they receive the emergency status, they follow predefined emergency strategy.

The next step is to refine the conceptual architecture into process and deployment architectures. The bottom portion of Fig. 8.8 shows the process architecture for the *Call Handler* component of the conceptual architecture. During the

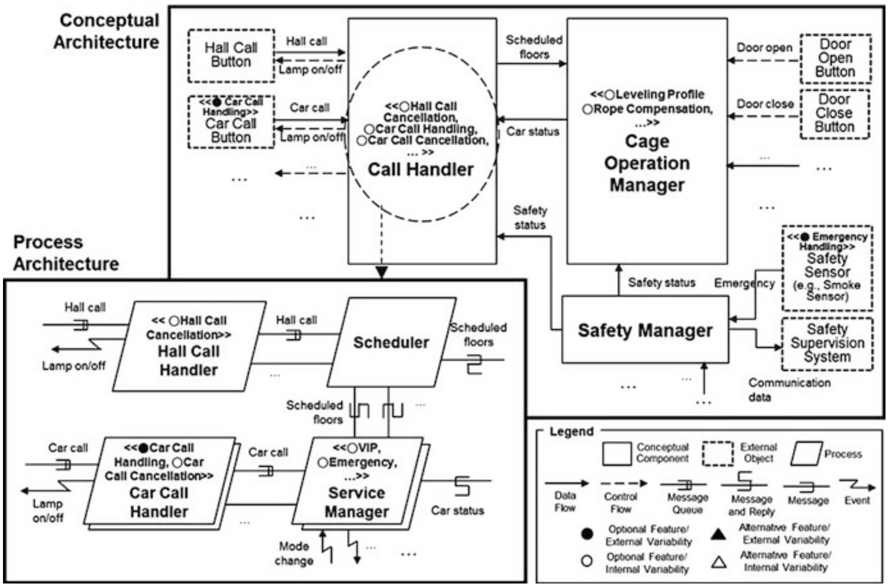

Conceptual Architecture
Process Architecture
Hall Call Button
Hall call
Lamp on/off
<<●Car Call Handling>> Car Call Button
Car call
<<○Hall Call Cancellation, ○Car Call Handling, ○Car Call Cancellation, ... >> Call Handler
Scheduled floors
Car status
Safety status
<<○Leveling Profile ○Rope Compensation, ...>> Cage Operation Manager
Door open
Door Open Button
Door close
Door Close Button
<<● Emergency Handling>> Safety Sensor (e.g., Smoke Sensor)
Safety status
Emergency
Safety Manager
Safety Supervision System
Communication data
<< ○Hall Call Cancellation>> Hall Call Handler
Scheduler
<<●Car Call Handling, ○Car Call Cancellation>> Car Call Handler
<<○VIP, ○Emergency, ...>> Service Manager
Mode change
Lamp on/off
Legend
Conceptual Component
External Object
Process
Data Flow
Control Flow
Message Queue
Message and Reply
Message
Event
● Optional Feature/ External Variability
○ Optional Feature/ Internal Variability
▲ Alternative Feature/ External Variability
△ Alternative Feature/ Internal Variability

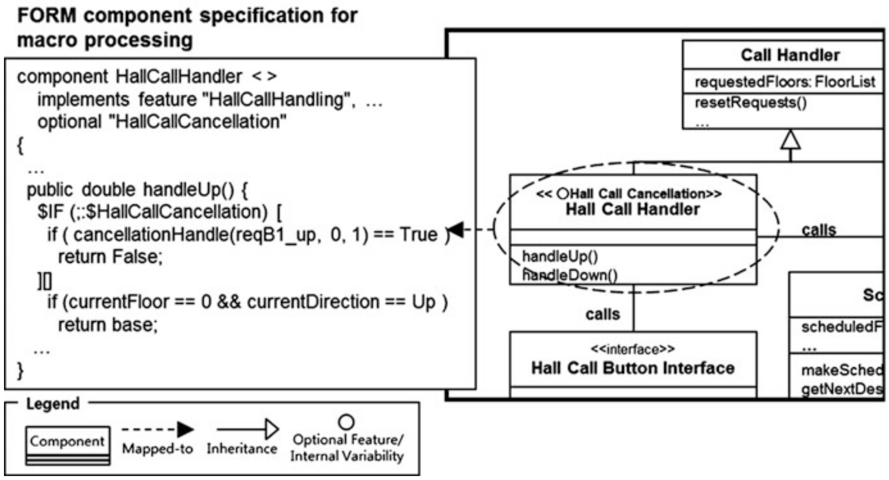
Fig. 8.8 Architecture design and refinement for the high-rise office building ECS

refinement, the quality attributes from the MPP are used for architectural style selection and evaluation [16]. For example, “Independent Component” architectural style [16] is selected, and *Scheduler* is designed to schedule a group of elevators, while each instance of *Service Manager* is designed to control each individual elevator of the group so that efficiency elevator management is achieved (i.e., they addresses the efficiency requirement).

6.2 *Asset Implementation*

Once conceptual architectures are refined into process and deployment architectures, the architectural components are then refined into concrete components. The PL component design consists of specifications of components and relationships among them. Figure 8.9 includes the component specification of the *Hall Call Handler* component and relationships with other components using UML.

For the component design, the product feature delivery methods in the MPP should be taken into consideration. For example, the use of FORM macro language (i.e., `$IF(;$HallCallCancellation)[-]`) in the component specification of the *Hall Call Handler* component in Fig. 8.9 is to support the prepackaged feature delivery method of the general hospital ECS. When the *Hall Call Cancellation* feature (in Fig. 8.5) is selected as one of prepackaged features in the general



```
component HallCallHandler < >
   implements feature "HallCallHandling", ...
   optional "HallCallCancellation"
{
 ...
 public double handleUp() {
   $IF (;:$HallCallCancellation) [
     if ( cancellationHandle(reqB1_up, 0, 1) == True )
       return False;
   ]]
     if (currentFloor == 0 && currentDirection == Up )
       return base;
  ...
}
```

Fig. 8.9 Component specification for Hall Call Handler

hospital ECS, code segments related to the *Hall Call Cancellation* feature are incorporated into the product at product build time.

Depending on the nature of extensions required for product specific features, techniques such as code generation, encapsulation, parameterization, framework, template, etc. may be used. For example, the *Service Manager* component in Fig. 8.8 that encapsulates various elevator control policies (e.g., VIP driving, Emergency driving) may be specified using a formal specification technique (e.g., Statechart specification), from which the program code may be generated after formal verification. Whenever new features are added, the feature interaction specification is modified and tested for correctness and new updated program code for the component is generated.

7 Discussion

In order to develop reusable core assets for a product line, PL software engineering must have an ability to exploit commonality and manage variability. A feature-oriented approach to commonality and variability analysis has been used for PL software engineering both in industry and academia, since the idea of feature-oriented analysis (i.e., Feature-Oriented Domain Analysis (FODA) [8]) was first introduced in 1990 by the Software Engineering Institute.

There are several reasons why FODA 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 PL asset development. The second reason is that FODA is an effective way to identify variability (and commonality) among different products in a PL. 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. The last reason is that the feature model can provide a basis for developing, parameterizing, and configuring various reusable assets (e.g., PL requirement models, PL architectures, 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 PL.

Notice that we have not discussed variability in product engineering. It is our opinion that variability of product line must be managed through product line asset engineering; otherwise there will be a proliferation of product versions. Features that are specific only to a product also need to be incorporated into the product through the variability mechanism of the asset management to avoid reworks that may happen when other features of the product evolve.

8 Summary and Outlook

We have explored variability issues over the product line life cycle starting from market analysis through component design to product instantiation. The marketing issues must be studied thoroughly and engineering feasibility of the envisioned products must be analyzed carefully before PL asset development starts.

One of the most important aspects of variability management is managing the explicit horizontal and vertical connections among PL assets including MPP, architecture, and components using various variability models and maintaining their consistencies. Here, the horizontal connection means variability relationships between PL lifecycle products such as MPP, PL architecture, and components, while the vertical connection means relationships between elements of each lifecycle product.

With this connection, marketing, which has traditionally focused only on securing and expanding the market share and on sales strategies, is forced to become more "product aware" and think about how features will be packaged, delivered, and maintained, who will perform these activities, and what the pricing implications are with various alternative approaches. This marketing-oriented perspective can be very effective in uncovering critical quality attributes required for product line architecture and component design. By tightly coupling the marketing with asset development, we can develop PL assets that will support the business goals and satisfy the needs of customers.

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

References

1. Clements, P., Northrop, L.: Software Product Lines: Practices and Patterns. Addison-Wesley, Boston, MA (2001)
2. Weiss, D.M., Lai, C.T.R.: Software Product-Line Engineering: A Family-Based Software Development Process. Addison Wesley, Boston, MA (1999)
3. Bosch, J.: Design and Use of Software Architectures: Adopting and Evolving a Product-Line Approach. Addison-Wesley, Boston, MA (2000)
4. Kang, K., Donohoe, P., Koh, E., Lee, K., Lee, J.: Using a marketing and product plan as the key design driver for product line asset development. In: 2nd International Software Product Line Conference, pp. 19–22 (2002)
5. Chastek, G., Donohoe, P., Kang, K., Thiel, S.: Product line analysis: a practical introduction. Technical report CMU/SEI-2001-TR-001, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA (2001)
6. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, Boston, MA (1995)
7. Kang, K.C., Cohen, S.G., Hess, J.A., Novak, W.E., Peterson, A.S.: Feature-oriented domain analysis (FODA) feasibility study. Technical report CMU/SEI-90-TR-21, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, November 1990
8. Czarnecki, K., Eisenecker, U.: Generative Programming: Methods, Tools, and Applications. Addison-Wesley, Boston, MA (2000)
9. Simos, M., et al.: Software technology for adaptable reliable systems (STARS). Organization domain modeling (ODM) guidebook, version 2.0. Technical report, STARS-VC-A025/001/00, Lockheed Martin Tactical Defense Systems, Manassas, VA (1996)
10. Lee, K., Kang, K.C., Lee, J.: Concepts and guidelines of feature modeling for product line software engineering. In: Gacek, C. (ed.) ICSR-7. LNCS, vol. 2319, pp. 62–77 (2002)
11. 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 (2010)
12. Lee, H., Choi, H., Kang, K.C., Kim, D., Lee, Z.: Experience report on using a domain model-based extractive approach to software product line asset development. In: Edwards, S.H., Kulczycki, G. (eds.) ICSR 2009. LNCS, vol. 5791, pp. 137–149 (2009)
13. Gomaa, H.: Designing Software Product Lines with UML: From Use Cases to Pattern-Based Software Architectures. Addison-Wesley, Boston, MA (2005)
14. 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)
15. Bass, L., Clements, P., Kazman, R.: Software Architecture in Practice. Addison-Wesley, Boston, MA (1998)
16. Kang, K.C., Lee, K., Lee, J., Kim, S.: Feature oriented product line software engineering: principles and guidelines. In: Itoh, K., Kumagai, S., Hirota, T. (eds.) Domain Oriented Systems Development: Practices and Perspectives. Gordon Breach Science, UK (2002)