Visualizing Product Line Domain Variability by Aspect-Oriented Modeling

Reinhard Stoiber, Silvio Meier and Martin Glinz
Department of Informatics, University of Zurich, Switzerland
{ stoiber | smeier | glinz } @ ifi.uzh.ch

Abstract

Modeling variability is a core problem in software product line engineering. The relationship between variability and commonality in a software product line bears strong similarities to the relationship between crosscutting concerns and core concerns in aspect-oriented modeling. So modeling variability with aspect-oriented techniques is an obvious idea which has been exploited before to some extent.

In this paper, we propose a new approach to modeling and visualizing variability by a combination of aspect-oriented variability modeling with table-based modeling of configuration possibilities and constraints. As a modeling language, we use a slightly extended version of the ADORA language.

Our main contributions are a visual, integrated model comprising both the commonality and the variability of the product line and a novel mechanism for synthesizing products from this model based on the aspect weaving capabilities of ADORA.

1. Introduction

Software product line engineering has gained a broad interest in academia as well as in industry over the past decade. A software product line [3] is a family of software applications in a common application domain, sharing a set of common features. The given variety of different applications is specified by the product line variability. When developing such application families, software product line engineering increases the overall product quality and customer satisfaction and at the same time decreases cost and time for development [17].

Although not all product line engineering approaches explicitly address a product line requirements document, it is advantageous to have one [5]. Such a document expresses the variability by specifying variation points and product specific variants that can be bound to the variation points [18].

However, current modeling languages such as UML [15] do not support modeling of variability. So the modeler has two alternatives which are both unsatisfactory: Either the commonalities and all variants are represented in a single model. This means that variants must be identified manually without support for modeling constraints on variants or dependencies between them, which leads to inaccurate and erroneous models. Or the modeler creates a separate model for each product variant. This means the common requirements must be replicated in every product variant, creating redundancy and, as a consequence, inefficiency and potential inconsistency.

In this paper we propose a new software variability modeling approach, building on our experience in modeling aspects in requirements and architecture models. By modeling variability with modularized crosscutting concerns, i.e. aspects, by employing a decision model to manage the variability concerns, and by augmenting our aspectual join relationship semantics to accord with the decision model, we solve the accuracy, efficiency and consistency problems. By modeling all variabilities and commonalities in one common, integrated domain model, we also achieve better understandable and maintainable product line requirements models. Moreover, profiting from our achievement in weaving aspect-oriented models [11], we can support automatic product derivation, when building and negotiating single products.

The remainder of the paper is organized as follows: in Section 2 we describe existing conventional variability modeling techniques. In Section 3 we briefly introduce aspect-orientation and other work on aspects and variability. Section 4 describes our approach and gives an example. Section 5 concludes the paper with a discussion and planned future work.

2. Conventional Variability Modeling

Research and practice in product line requirements engineering brought up many approaches and tech-

niques in the last decade. Probably the most widely used technique is feature-oriented domain analysis (FODA) [8]. Using features to represent high-level, customer-relevant characteristics of the product line is an intuitive approach and facilitates the communication of common and variable requirements. FODA typically supports common, optional and alternative features to introduce variability. In the past, it has been widely used in product line practice and also as a basis in research. Feature trees can be enriched in different ways, for example, by adding rationale and constraints to assure valid feature configurations [1]. However, even though the method is successful, feature models can still accomplish only a part of the requirements engineering. They are well suited for customer negotiation, thus facilitating the first requirements tasks, but they are not sufficient to build entire requirements models.

In requirements engineering, UML [15] is often regarded as the *de-facto* modeling language for conventional systems. UML supports many modeling notations for different views on the requirements and architecture level. Gomaa et al. [7] developed an UML-based research prototype providing a tool framework for software product line engineering and product derivation. They model use cases, collaborations, classes, statecharts, features and multiple product line views. They use the feature model as a unifying view to support feature-based product derivation. Their approach supports automatic consistency checking between the different views and models. In summary, they combine many separate notations to model the requirements and realize the variability management with feature models as a unifying view.

Pohl et al. [17] use a similar, UML-based approach. They propose orthogonal variability modeling, providing a general variability model comprising the domain's variation points and variants. For requirements modeling they use notations like text, features, use case models, data flow diagrams, class diagrams or state machine diagrams. All these models include the commonality and variability. Separately, they develop an orthogonal variability model of the domain and link the variation points and variants between the orthogonal model and the concrete correspondents in the requirements engineering artifacts. In this way, they can use traditional requirements engineering also for product lines, but nevertheless identify and trace variability and commonality in requirements. Compared to [7], Pohl et al. employ explicit orthogonal variability models for variability management, instead of feature modeling. Both approaches use multiple separate notations to describe the requirements.

Another related approach is Schmid et al. [19]. They present a customizable approach for variability management that can be applied to company-specific modeling notations. Central to this method is the decision model, in which the entire domain variability is documented. Schmid et al. present cookbook-like procedures to enhance specific modeling notations with a set of variability selection types. These types include optionality, set optionality, alternative, set alternative and value reference selection. This makes the approach customizable and applicable to every software lifecycle phase. The weakness of this approach, however, is that it will hardly be possible to use existing tool support since they a priori build on specific, non-standard notations. This approach has been successfully applied in large-scale product lines in the European industry [4].

Software architecture is another major field in variability modeling. There exist many architecture description languages (ADLs) in different application areas [10]. In the field of software product lines, two prominent representatives are xADL 2.0 [21] and Koala [16]. ADLs naturally focus on architecture, so they cannot replace traditional requirements engineering. However, xADL 2.0 for example possesses quite advanced variability management techniques, which can also be interesting for requirements.

3. Aspect-Oriented Modeling

3.1. Aspect-Oriented Software Development

Aspect-Oriented Software Development (AOSD) deals with so-called crosscutting concerns in software systems. These cannot be separated from other concerns by conventional modularization means for software artifacts. A crosscutting concern impacts other software artifacts and the impacted artifacts cannot control the way they are impacted, as argued in [11]. This leads to effects such as scattering and tangling of the crosscutting concerns over other non-crosscutting ones. AOSD aims at introducing ways to modularize crosscutting concerns and denoting their relationship to other concerns in the software process [2]. An early separation and modularization of cross-cutting concerns further improves the product quality and reduces the adjustment, maintenance and evolution cost. Apart from these advantages, the use of aspect-oriented techniques to visualize requirements models may help to better understand the models [11] [12].

3.2. Aspect-Orientation and Variability

There is various work emphasizing the commonalities between AOSD and software product line engineering, e.g. [9] [20] [14] [13]. Software variability impacts a software system in a similar way as crosscutting concerns do, while commonalities behave like non-crosscutting concerns. In this way we have a correspondence between commonality and core concerns on the one hand and between aspects (representing crosscutting concerns) and variability on the other hand. Both aspects and variability are orthogonal concepts which are independent of the core system (core concerns / commonality) and can freely be combined with it.

Furthermore, software variability may impact commonalities as well as other software variabilities. In the same way, crosscutting concerns (aspects) may impact non-crosscutting ones as well as other crosscutting concerns. Therefore, aspect-oriented techniques can also be used for modeling the variable and the common concerns in a software product line separately. For example, Loughran et al. [9] use aspect-oriented techniques together with natural language and concern identification for the derivation of suitable feature-oriented models for implementation. Siy et al. [20] present an approach for an aspect-oriented description of product line requirements by handling system requirements, exception handling requirements (alternate flows) and non-functional requirements as aspects in their framework. Their approach is based on textual requirements specification and is strongly dependent on the use of requirement tags for aspects to locate where to weave in. The work by Mezini and Ostermann [13] further sheds light on the applicability of AOP beyond the traditional examples of logging, debugging, authorization control, and the like. They combine feature-oriented and aspect-oriented approaches with the result of gaining less code scattering of singular features by using traditional aspects also for variability. Consequently, they improve feature-oriented techniques by aspect-orientation. In [14], Nyssen et al. elaborate how the use of aspects helps with the realization of domain variability in feature models. Their result is that aspects complicate the readability and comprehension of architectures and that they won't recommend it for product lines. However, they didn't employ any means for visualization of the implicit communication links between aspects and common components.

4. Modeling Variability with Aspects

In this section, we introduce our approach, using a simple security system as an example. This system consists of sub-components for an alarm system, a monitoring system and a door opening system. The first two are not specified in more detail. The third, the electronic door opening system, consists of two different options: a fingerprint reader and a keypad plug-in. At least one of these two has to be chosen and also both can be included in a valid system. The keypad plug-in further offers two alternative variants of keypads: a hardware keypad and a touchscreen keypad. The keypad plug-in is in any case needed when selecting one keypad variant. This example represents a simple, partially described product line domain. The mentioned components can be understood as product features, describing customer-relevant functionality of the system.

As a modeling language, we use ADORA [6] with the aspect modeling extensions described in [11] [12]. Recall that conventional modeling languages such as UML have severe difficulties with handling and maintaining variability, as described in Section 1.

4.1. Modeling a Product Line Domain

When modeling the system, see Figure 1, we begin by modeling the commonality. We generate an abstract object for the electronic security system and embed the sub-components Alarm System, Monitoring System and Door Opening in it. For the first two components, we have hidden the details, indicated by the ellipsis after the name. The door opening system contains a use case Open the Door as a commonality and this use case is performed by an external actor whom we call House Owner.

Now we model the *variability*. For the door opening, there exist two options, a keypad and a fingerprint reader. The latter is either a hardware or a touchscreen keypad. Every variant is modeled as an ADORA aspect container (a rectangle with two cut-off edges) in Figure 1. Next, we model the details of the variants. In ADORA we describe the behavior and the user-interaction of a component with statecharts and so-called scenario-charts, respectively [6]. Rectangles with rounded edges denote states, ovals denote (type-level) scenarios, which constitute use cases or use case steps in UML terminology.

Modeling of variability semantics is also illustrated in Figure 1: The door opening by keypad and by fingerprint reader are two separate options, where the user

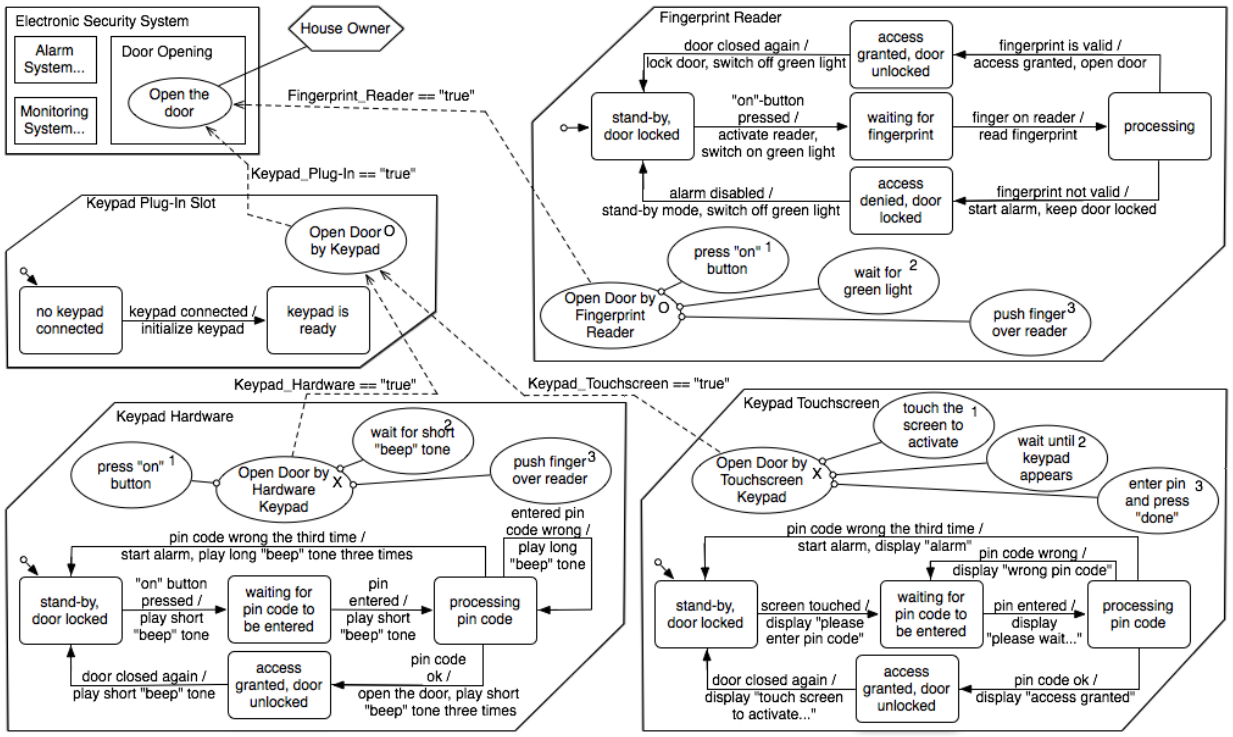

Figure 1. The electronic security system domain model with variability modeled by aspects and additional annotations, in the ADORA language.

Table 1. The decision model for the security system; an orthogonal representation of the domain variability.

Name	Relevance	Description	Range	Multiplicity	Constraints	Binding Time
Fingerprint_Reader		Is there a fingerprint scanner?	true, false	1	Fingerprint_Reader = false -> Keypad_Plug-In = true	Installation
Keypad_Plug-In		Is there a Keypad for a numeral code?	true, false	1	Keypad_Plug-In = false -> Fingerprint_Reader = true	Installation
Keypad_Hardware	Keypad_Plug-In == true	Is it a Hardware Keypad?	true, false	1	Keypad_ Touchscreen = true -> Keypad_HW = false	Installation
Keypad_Touchscreen	Keypad_Plug-In == true	Is it a Touchscreen Keypad?	true, false	1	Keypad_ HW = true -> Keypad_ Touchscreen = false	Installation

has to choose one; we model the two identifying variant scenarios with an "O", saying that at least one has to be chosen and also both are possible (logical or). Further, for the two alternatives of realizing the keypad, we model the identifying scenarios with an "X", to indicate that only one of those two is possible (logical exclusive-or).

The other important parts of our proposed variability modeling approach are the *join relationships*. These are annotated by logical terms, including decision variables, which have to be evaluated to true in order to weave in the variability represented in the aspect containers. To manage these decision variables, we introduce a decision model, as proposed in [19]; see Table 1. In the decision model, we describe all decision variables and their use in detail. Every decision variable is represented by one row in the table and has an identifying name, a condition for relevance, a verbal description of what it decides, a range of values it can take, a multiplicity defining how often it can exist, a constraint where we define all the conditions that have to be met when deciding the variable, and a binding time which defines the latest point in time for the decision to be taken.[1]

The *constraints* play an important role in the decision model: they precisely represent dependencies between different variability decisions by using *and*, *or*, *exclusive-or* operators, mutual exclusion and the like

[1] Further, a *recommendation* column could be added, providing a default value in situations where a modeler is not sure about how to decide.

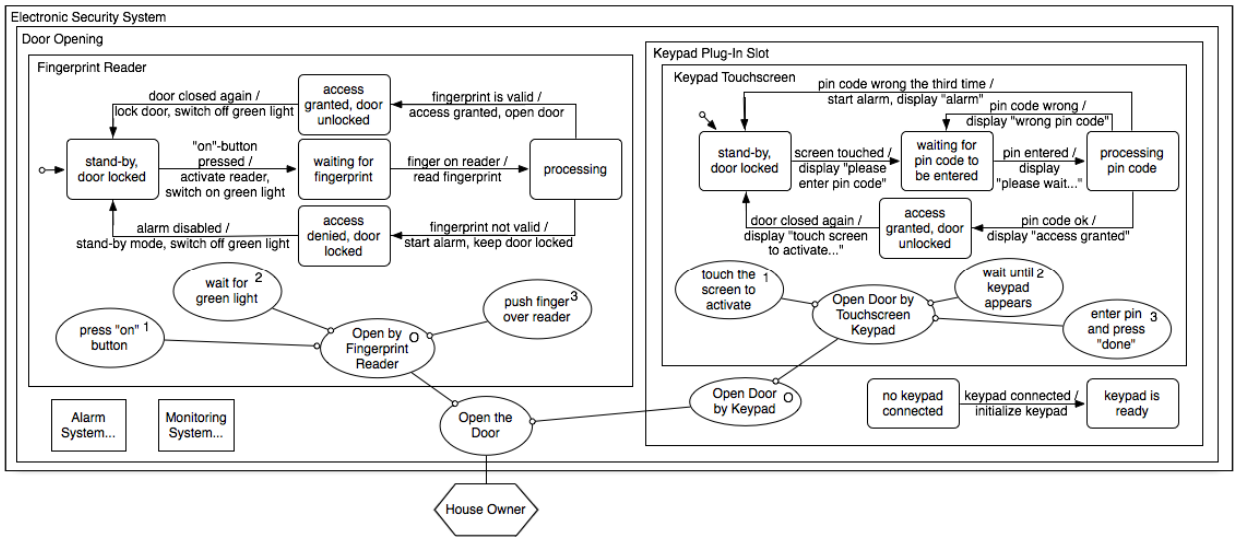

Figure 2. A possible electronic security system application model with resolved variability, in the ADORA language.

between different decision variables. In Table 1, for example, we modeled the facts that Fingerprint_Reader or Keypad_Plug-In (or both) must be selected and that either Keypad_Hardware or Keypad_Touchscreen must be selected (if the relevance condition is fulfilled) by constraints on the decision variables. The constraints must always be true when a decision variable is decided, to provide valid configurations. The relevance condition additionally qualifies a decision for product derivation; if it is not true, the decision does not need to be taken and the constraint does not need to be fulfilled.

With the product line commonality and variability model (Figure 1) and the additional decision model for orthogonal description of the detailed variability relationships (Table 1), we can describe software product line domain requirements to a considerable extent.

4.2. Deriving Applications from the Product Line Domain

We now describe the derivation of an application from the product line domain, again using our example.

As a basis, we use a weaving mechanism for aspects already provided by ADORA [11]. By reusing and partly expanding this mechanism, we can achieve an *automatic product derivation* for application engineering. This means that during product derivation, when negotiating with customers, engineers can take decisions on variability and instantly visualize the resulting, partly or fully derived software application model.

To generate a valid software application, we consider the values "true" to be taken for all decisions except the touchscreen keypad. By employing an automatic product derivation, the resulting derived application looks as shown in Figure 2.

Figure 2 represents an application example where all domain variability is already resolved. With our approach, also partial variability selection is possible. For example, the fingerprint reader and the keypad plug-in variabilities might already be selected and woven into the model, while the two keypad subvariants might still be displayed as alternative variants, represented by aspects.

5. Discussion

By building our approach upon the modeling language ADORA, we benefit from a full range of already existing features of the language and tool prototype. These benefits include partial modeling, zooming into and out of detailed model descriptions, the possibility to hide partial views of the model and intelligent model visualization algorithms for presenting the model in an appealing form [6]. Furthermore, we can also benefit from the traditional ADORA aspect modeling capabilities if we have, for example, crosscutting concerns within a variant.

Our approach is rather heavy-weight compared to feature-oriented methods. This can be a disadvantage, especially for customers to understand the models in the early phases of the product line requirements negotiations. Another potential weakness is the complexity

of the visual models, which is further increased by expanding the language with variability modeling techniques. This may make it harder to understand these models compared to single system requirements models. The fact that our notation distinguishes variability from conventional aspects only by join relationship annotations may also be a problem in practice – this needs to be investigated further.

With our aspect-oriented variability modeling approach we only demonstrated so far how to "connect" variability with commonality by use cases. There are also software systems representing only behavior. To handle such cases we still need to define further language semantics. For example, certain situations demand to weave in only partial scenarios and/or behavior, as can be realized with conventional aspects. For such cases our approach needs to be refined.

For future research we plan to define a complete semantics to basically enable the requirements modeling language ADORA for product line analysis. To comprehensively support software product line domain modeling, we will also need to find an appropriate solution to integrate the decision model within the ADORA tool prototype and to handle variability constraints within our models. Validation and verification of the domain and application models will also be an important issue. For the implementation of the automatic product derivation we will build on our experience in realizing aspect-oriented weaving.

6. References

[1] Asikainen, T.: Modelling Methods for Managing Variability of Configurable Software Product Families. Licentiate Thesis, Helsinki University of Technology. 2004.

[2] Chitchyan, R., Rashid, A., Sawyer, P., Bakker, J., Alarcon, M. P., Garcia, A., Tekinerdogan, B., Clarke, S., and Jackson, A.: Survey of Aspect-Oriented Analysis and Design. In R. Chitchyan, A. Rashid (eds.): AOSD-Europe Project Deliverable No. AOSD-Europe-ULANC-9., 2005.

[3] Clements, P., Northrop, L.: *Software Product Lines: Practices and Patterns.* Edison Wesley, 2001.

[4] Dhungana, D., Rabiser, R., Grünbacher, P., Lehner, K., Federspiel, C.: DOPLER: An Adaptable Tool Suite for Product Line Engineering. *11th International Software Product Line Conference (SPLC 2007)*. Kyoto, Japan. 2007.

[5] Faulk, Stuart R.: "Product-Line Requirements Specification (PRS): An Approach and Case Study". *Fifth IEEE International Symposium on Requirements Engineering (RE'01).* 2001.

[6] Glinz, M., Berner, S., and Joos, S.: Object-Oriented Modeling with ADORA. *Information Systems*, 27 (6). 2002.

[7] Gomaa, H., Shin, M. E.: Automated Software Product Line Engineering and Product Derivation. *40th Hawaii International Conference on Software Systems.* 2007.

[8] Kang, K., Cohen, S., Hess, J., Novak, W., Peterson, A.: Feature Oriented Domain Analysis Feasibility Study", SEI Technical Report CMU/SEI-90-TR21 1990.

[9] Loughran, N., Sampaio, A., Rashid, A.: From Requirements Documents to Feature Models for Aspect Oriented Product Line Implementation. *Workshop on MDD in Product Lines at MODELS 2005*. 2005.

[10] Medvidovic, N., Taylor, R. N.: A Classification and Comparison Framework for Software Architecture Description Languages. In: *IEEE Transactions of Software Engineering*, Vol. 26, No. 1, January 2000.

[11] Meier, S., Reinhard, T., Stoiber, R., Glinz M.: Modeling and Evolving Crosscutting Concerns in ADORA. *11th Workshop on Early Aspects at ICSE'07*. Minneapolis, USA. 2007.

[12] Meier, S., Reinhard, T., Seybold, C., Glinz, M.: Aspect-Oriented Modeling with Integrated Object Models. *Modellierung 2006*. Innsbruck, Austria. 2006.

[13] Mezini, M. and Ostermann, K.: Variability management with feature-oriented programming and aspects. *12th ACM SIGSOFT International Symposium on Foundations of Software Engineering.* Newport Beach, USA. 2004.

[14] Nyssen, A., Tyszberowicz, S., and Weiler, T.: Are aspects useful for managing variability in software product lines? A case study. *Aspects and Software Product Lines: An Early Aspects Workshop*, at SPLC-Europe'05. 2005.

[15] Object Management Group - UML: http://www.uml.org/

[16] van Ommering, R., Van der Linden, F., Kramer, J., Magee, J.: The Koala Component Model for Consumer Electronics Software. *IEEE Computer*, 33(3): 78-85. 2000.

[17] Pohl, K.; Böckle, G.; van der Linden, F.: *Software Product Line Engineering – Foundations, Principles, and Techniques.* Springer, Heidelberg 2005.

[18] SEI/CMU Software Product Lines Homepage: http://www.sei.cmu.edu/productlines/ 2007.

[19] Schmid, K.; John, I.; "A customizable approach to full lifecycle variability management", *Science of Computer Programming*, Vol. 53, No. 3, Elsevier, December 2004.

[20] Siy, H., Aryal, P., Winter, V., and Zand, M. 2007. Aspectual Support for Specifying Requirements in Software Product Lines. *Workshop on Early Aspects at ICSE*. Minneapolis, USA. 2007.

[21] xADL 2.0 – A Highly Extensible Architecture Description Language for Software and Systems: http://www.isr.uci.edu/projects/xarchuci/