On the Interdependence and Integration of Variability and Architectural Decisions

Ioanna Lytra
Software Architecture Research Group
University of Vienna, AT
ioanna.lytra@univie.ac.at

Holger Eichelberger
Institute of Computer Science
University of Hildesheim, DE
eichelberger@sse.uni-hildesheim.de

Huy Tran
Software Architecture Research Group
University of Vienna, AT
huy.tran@univie.ac.at

Georg Leyh
Siemens AG
Erlangen, DE
georg.leyh@siemens.com

Klaus Schmid
Institute of Computer Science
University of Hildesheim, DE
schmid@sse.uni-hildesheim.de

Uwe Zdun
Software Architecture Research Group
University of Vienna, AT
uwe.zdun@univie.ac.at

ABSTRACT

In software product line engineering, the design of assets for reuse and the derivation of software products entails low-level and high-level decision making. In this process, two major types of decisions must be addressed: variability decisions, i.e., decisions made as part of variability management, and architectural decisions, i.e., fundamental decisions to be made during the design of the architecture of the product line or the products. In practice, variability decisions often overlap with or influence architectural decisions. For instance, resolving a variability may enable or prevent some architectural options. This inherent interdependence has not been explicitly and systematically targeted in the literature, and therefore, is mainly resolved in an ad hoc and informal manner today. In this paper, we discuss possible ways how variability and architectural decisions interact, as well as their management and integration in a systematic manner. We demonstrate the integration between the two types of decisions in a motivating case and leverage existing tools for implementing our proposal.

Categories and Subject Descriptors

D.2.11 [**Software Architectures**]: Domain-specific architectures; D.2.13 [**Reusable Software**]: Reuse Models

Keywords

Variability Decisions, Architectural Decisions, Software Product Lines, Product Derivation

1. INTRODUCTION

Variability management and architecture-centric development are fundamental aspects of software product line engineering [9]. Variability management aims at the explicit modeling of differences (variabilities) among the products that can be derived from a product line and, in particular, the interdependencies among individual variabilities. From a variability management perspective, the software architecture of a product line describes the design of all products in a product line in terms of reusable assets. This requires, first of all, that commonalities and variabilities among the different products of a given product line are identified. The aim of software product line engineering is to create a single architecture for a range of related products that can be tailored and customized to meet the requirements of the derivable products, for instance, imposed by different customers. This architecture is often called the *reference architecture* of the product line and may contain variabilities to represent the differences among the products [9]. Finally, each product has its own architecture derived from the reference architecture.

From an architectural perspective, variabilities may reflect different architectural options considered during the design of the product line that are independent of the products' features. Bachmann and Bass pointed out two causes of variability in the software architecture of a product line: (1) product line architectures encompass a collection of different alternatives that must be resolved during product configuration, and (2) at design time multiple alternatives may exist and need to be captured [1].

Currently, variability management and architectural design are mostly treated as separate activities. For their respective needs, product line and architecture communities use a variety of methods and tools for modeling, documenting, and making specific types of decisions. The product line community has mainly adopted feature models (e.g., [8]) and decision models (e.g., [14]) to specify variability. The software architecture community has exploited techniques from variability modeling (such as COVAMOF [18]) and used architectural decision modeling to describe variabilities and connect them to quality attributes [20]. Existing variability management approaches focus on describing variabilities in a product line and managing their impact on the derived products. On the other hand, existing architectural design and decision support tools [17] cover various architectural aspects, views, and reasoning of decisions but lack adequate support for interaction with variability decisions. To the best of our knowledge, the interdependence and integration of variability and architectural decisions have neither been studied nor addressed in a systematic way, yet. This work intends to fill this gap and proposes the systematic integration of the two types of decisions.

Based on a motivating case, we discuss the interdependence of

VaMoS '14 January 22 - 24, 2014, Sophia Antipolis, France

http://dx.doi.org/10.1145/2556624.2556634.

variability decisions and architectural decisions in the development of the product line and the derived products. We propose to systematically integrate these two types of decisions and suggest integrated tool support based on two existing tools: ADvISE[1]—a tool for assisting architectural decision making—and EASy-Producer[2]—a tool for variability management.

The remainder of the paper is structured as follows. In Section 2, we briefly present the background for variability decisions and architectural decisions in product line engineering, as well as the implicit relations of both kinds of decisions documented in the literature. In Section 3, we introduce a motivating example from an industrial case study and discuss variability and architectural decision interdependencies in this context. We present our proposal in Section 4 and discuss its implementation in Section 5. In Section 6, we compare our approach to related work and, finally, in Section 7 we conclude and outline future work.

2. BACKGROUND

In this section, we provide background definitions that are relevant to this paper. We also provide a basic discussion of the dependencies between variability decisions and architectural decisions.

2.1 Product Line Engineering

To understand product line engineering it is important to notice a fundamental characteristic: the separation between development at the level of the product line as a whole and the level of an individual product; namely the *domain engineering* and the *application engineering*, respectively. We will use the simplifying terms *product line level* and *product level* though in this paper. At the product line level, all engineering activities that are relevant to a range of products—the product line—are performed. The logically first step is to determine what variability needs to be supported by the product line as a whole. As a basis for this, the scoping activity identifies which variability should actually be supported in a reusable manner [14]. Together with domain analysis, a precise model of the product line is derived from this. It should be noted that this excludes product-specific parts from further consideration. Variability decisions are captured in a variability (decision) model, which represents all variabilities (differences among the individual products) at an abstract level, as well as constraints among the individual variabilities. In later stages, the variability model is used to derive a valid product configuration for instantiation.

The reference architecture defines the realization of the product line, i.e., any decisions made in the reference architecture will be available in all products. The reference architecture may also contain variability in the sense that some architectural decisions are not finally taken for the product line as a whole, but several resolutions remain possible for the different variants. Thus, the fundamental potential architectural decisions are determined at the product line level. Besides, some architectural decisions may also be introduced as part of product-specific parts of the system.

It may also happen at the product level that product requirements are not fully covered by the product line. Thus, one of two situations may occur: (a) the additional requirements can be covered as product-specific functionality as they interfere only little with the parts covered by the product line, and (b) there is a strong impact (e.g., the change relates to some variability, but would require a variability option which is not available). In the latter case, it is necessary to re-evaluate the existing variability. This may lead to introducing new variabilities to the variability model; likewise, the architecture needs to be extended.

2.2 Variability and Architectural Decisions

When creating reference architectures, a large number of decisions must be made, such as how to model a certain variability or which design pattern provides adequate support for covering particular variabilities. Other decisions are left open until product derivation time. We refer to the decisions taken as part of variability management activities as *variability decisions*. Further, we will call the decisions related to the software design of the architectures of the product line and the products *architectural decisions*.

Conceptually, variability decisions and architectural decisions may pose distinctive as well as overlapping information. Variability decisions are any decisions that describe differences among different products in a product line and are relevant to reuse (i.e., excluding product-specific aspects). Typically, variabilities are described in terms of optional (yes-no), alternative (one-out-of-many), or multiple (many-out-of-many) selections [9]. An architectural decision is the result of the evaluation of alternative design options in terms of architectural elements such as patterns, components, or connectors and the selection of the best-fitting solution and are considered both at product line and product level. At product line level architectural decisions can also be postponed to the product level. These open decisions become variabilities.

Architectural decisions at different levels of granularity are usually taken in the early stages of design. Later, throughout the software development process, as well as the evolution of a product line, new architectural decisions may be considered and existing decisions might have to be revised. However, this is not a one-time process, as both engineering activities at the product line level may provide feedback to requirements engineering or variability management [4]. In particular, the variability supported by the product line has to be re-examined whenever new products are developed and introduce the need for new variabilities. A variability in a system can be implicit (present at higher levels of abstraction), designed (explicit) and bound (to a particular variant) [19].

In general, there are two potential options for dealing with such feedback from later stages to variability modeling: (a) by using a single variability model that captures variability on all levels in a homogeneous way, and (b) by using a so-called staged configuration approach in which multiple models are used, and information in one model is used to configure the subsequent level [5]. In practice—especially in commercial tools—usually an approach with a single central variability model is used to simplify management and development.

3. MOTIVATING EXAMPLE

As a motivating example for the proposed approach, we leverage a software product line for warehouse management systems. Based on the product line, custom-made software products for specific warehouses can be derived. The product line targets automated warehouses only; that is, goods in a warehouse are moved on pallets by conveyors or stapler cranes. Usually, a warehouse management system is accompanied by an Enterprise Resource Planning (ERP) system that handles all financial aspects of warehouse transactions and a base automation system that directly controls the conveyors and stapler cranes. This overall system architecture is typically layered, consisting of a Resource Planning Layer, a Warehouse Management Layer and a Basic Automation Layer (see Figure 1).

The most important business process of a warehouse is "Order Processing" as illustrated in Figure 2. The process is triggered

[1] http://swa.univie.ac.at/Architectural_Design_Decision_Support_Framework_(ADvISE)

[2] http://sse.uni-hildesheim.de/en/fb4/institutes/ifi/software-systems-engineering-sse/research/projects/easy-producer/

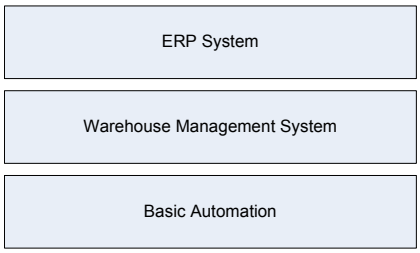


Figure 1: Architecture of a warehouse management system

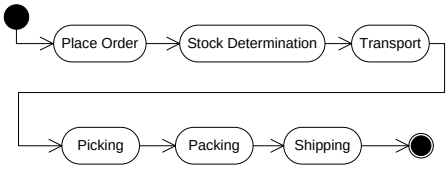


Figure 2: Goods out process of a warehouse

when a client orders some goods stored in the warehouse. Next, the ERP system notifies the warehouse management system about what shall be delivered ("Place Order"). The warehouse management system then maps the orders to boxes of goods that are stored in the warehouse management system ("Stock Determination"). Transport orders are sent to the base automation. The base automation will transport the boxes to a picking station ("Transport"). There, a human worker picks up the goods that are specified by the order ("Picking"). Finally, the goods are packed ("Packing") and sent to the customer ("Shipping").

3.1 Variability Decisions

To create a product line of warehouse management systems, the variability of the domain must be managed. A selected set of variabilities (variability decisions, possible values, and binding times) that we will use throughout this paper as running example is summarized in Table 1. The binding time is the latest point in the lifecycle when a variability decision at product level must be made. One of the variabilities we consider is the scale of the warehouse that can range from *high* (e.g., handling thousands of orders per day), *medium* (e.g., handling hundreds orders per day), and *low* (e.g., handling few dozen orders per day). Another variability concerns the strategy for handling partial pallet quantities that considers either speed or optimal reduction. The *high speed* strategy tries to only pick from a single box whilst it may possibly leave a lot of partial pallet quantities, while the *optimal reduction strategy* will remove as much partial pallet quantities as possible (thereby creating space in the warehouse), which leads to a higher amount of picks. The third variability represents different strategies for stapler cranes with one or several forks. The forth variability investigated in this example captures the user interface (UI) options including support for desktop/laptop computers or mobile devices.

Table 1: Selected variability decisions and their values in the warehouse product line

ID	Variability Decision	Possible Values	Binding Time
VP1	Picking rate	High/Medium/Low	Design time
VP2	Partial pallet strategy	High speed/Optimal reduction	Runtime
VP3	Stapler crane strategy	Single fork/Multiple forks	Design time
VP4	UI device	Computer/Mobile device	Design time

3.2 Architectural Decisions

Table 2 presents a subset of the design space under consideration that provides the basis for making architectural decisions at product line level. In our motivating case, we prefer an asynchronous call-oriented to a message-oriented interaction style because it leads to less complex and more readable code (cf. **AD1**); we prefer fix to variant interprocess communication (IPC) because fix IPC provides higher performance (cf. **AD2**); and we prefer a service-oriented to resource-oriented API style as the underlying infrastructure functionality is already provided in terms of services (cf. **AD3**).

Table 2: Exemplary architectural decisions at product line level

ID	Decision Point	Options
AD1	Interaction Style	Asynchronous calls interaction Message-oriented interaction Synchronous calls interaction
AD2	Interprocess Communication (IPC)	Fix Variant
AD3	API Style	Service-oriented Object-oriented Resource-oriented

Exemplary architectural decisions at the product level are shown in Table 3 along with the related variabilities. Some architectural decisions are influenced by the variabilities identified previously. For instance, we cannot select a single interprocess communication solution for **AD4** because of **VP1**. For *low* picking rate, the option *IPC open source* is sufficient, while for *medium* and *high* picking rates, *IPC very expensive* is necessary. This decision brings us to other subsequent architectural decisions. For instance, we need to decide if we will provide an abstraction interface for IPC invocations (cf. **AD5**) and depending on decision **AD4** we will create a wrapper component for IPC or even adapt its interface in order to support **VP1** (cf. **AD6** and **AD7**). Similar to the decision for an IPC solution, the deployment cannot be decided until **VP1** is chosen. A single server is necessary for *low* picking rates but multiple servers with a round-robin strategy should be used with respect to *medium* picking rates. For *high* picking rates, multiple servers with load monitoring are needed. In our example, we decided to always use a client side business delegate to identify the server, knowing that it is not necessary, and therefore, costly for single server deployment, in order to limit the variability of the architecture (cf. **AD9**). In summary, the aforementioned architectural decisions are related to and influenced by the variation point **VP1** as follows:

- Low picking rate implies *IPC open source* (**AD4**) and *Single server* (**AD8**)
- Medium picking rate implies *IPC very expensive* (**AD4**) and *Multiple server with round-robin* (**AD8**)
- High picking rate implies *IPC very expensive* (**AD4**) and *Multiple server with load monitoring* (**AD8**)

4. PROPOSED APPROACH

In this section, we introduce our approach for integrating variability and architectural decisions in a systematic manner. Our approach is presented in the context of both variability and architectural decision making processes at product line and product level for designing reference architectures and product architectures respectively. In particular, we present the steps of variability management and architectural decision making and their relationships along with our solutions for eliciting the interdependencies among the two kinds of decisions.

Table 3: Exemplary architectural decisions at product level

ID	Decision Point	Options	Variability Decision
AD4	IPC	IPC open source	VP1-low
		IPC medium price	-
		IPC very expensive	VP1-medium/high
AD5	IPC invocations	No abstraction interface	-
		Abstraction interface(facade)	-
		Abstraction interface(gateway)	-
AD6	IPC open source	Adapt IPC open source	-
		Create a wrapper component	-
AD7	IPC very expensive	Create a wrapper component	-
		Other	-
AD8	Deployment devices	Single server	VP1-low
		Multiple servers/round-robin	VP1-medium
		Multiple servers/load monitoring	VP1-high
AD9	Server identification	Business delegate proxy	-
		Business delegate adapter	-

4.1 Approach Overview

An overview of our approach is provided in Figure 3. First of all, the variability model at the product line level is derived from scoping and subsequent domain analysis. It leads through a manual process of derivation to a corresponding architectural decision model. As part of variability modeling, dependencies among variabilities are expressed using constraints, for instance, selecting a certain kind of warehouse restricts the range of applicable partial pallet handling strategies. To support the creation of the reference architecture, architectural decisions for the product line are identified in the architectural decision model. Our approach formally elicits the dependencies between the variabilities and architectural decisions in terms of mappings between the corresponding elements. At product line level, the resulting reference architecture will be designed to cover the whole range of variability specified by the variability decision model by selecting the appropriate architectural solutions from the architectural decision model.

At the product level, the variabilities of the product are resolved in order to obtain a valid configuration, i.e., all variability constraints are satisfied by the decisions made. Using the aforementioned formal mappings as input, we can automatically constraint the available architectural options that correspond to variability decisions made at product level. For example, if a specific variability option is chosen in the configuration, the resolution of the aforementioned predefined mappings ensure that only architectural options that are associated with that specific variability option can still be chosen in the architectural decision model. At this point, further architectural decisions can be made for the needs of the product architecture. This way, variability and architectural decisions are kept consistent to each other and the product architecture conforms both to the variability and architectural constraints. A reference of the variability and architectural decision models used in our approach can be found in [16] and [10] respectively.

4.2 Product Line Level

The key idea of our approach is to first determine necessary variability decisions based on the requirements, as well as possible architectural solutions for implementing the required variability (also architectural decisions not related to variability). Then, the possible variability resolutions (variants) are mapped to the corresponding architectural options. This is realized in the following steps:

S1. Identify Variabilities: Based on scoping and an analysis of the requirements, we identify potential variabilities. This is often done by determining main features that are relevant to specific system instances [14].

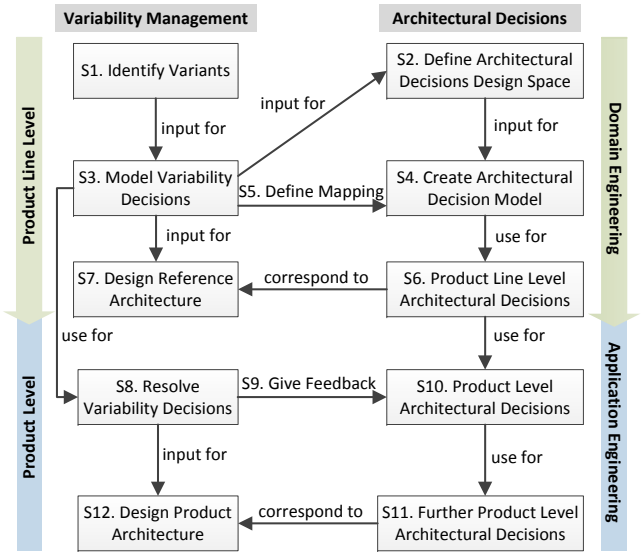


Figure 3: Approach overview

S2. Define Architectural Decisions Design Space: We consider existing architectural knowledge, such as reusable architectural models (e.g., [21]), to define the architectural decisions design space, i.e., architectural options and alternatives for the various decision points related to the design of the reference and product architectures. This information will be used as input for creating the architectural decision model.

S3. Model Variability Decisions: The individual variants that vary along an identifiable theme can be described as variability decisions. The advantage of having them as variability decisions—rather than using other variability modeling techniques such as feature modeling—is mostly that we make the inherent dimension of variability explicit[3].

S4. Create Architectural Decision Model: The analysis made in **S2** helps us define the architectural decision model that will be used as guidance for making architectural decisions at product line and product level[4].

S5. Define Mapping: The product line architects identify which variability decisions correspond to architecturally relevant requirements. They determine potential architectural decisions that correspond to the individual variants, and make this interdependencies explicit by introducing a mapping between the two models. For instance, a variability decision may exclude or enforce a related architectural decision.

S6. Product Line Level Architectural Decisions: The architectural decisions that will cover the desired variability are derived manually. The aim is to create a strategy that covers the whole range of variants described by a variability decision, considering the architectural alternatives and options provided by the architectural decision model. The architectural decisions at product line level are reflected in the reference architecture and

[3]Note that we will use a decision modeling approach [16] as the tool that we will discuss later (EASy-Producer) is based on this approach. However, decision modeling and feature modeling are rather similar today and can even be equivalent for some cases [7].

[4]Note, that we will apply decision modeling based on Questions, Options and Criteria [11] as the ADvISE-tool for decision making support used in our approach is based on this. However, other architectural decision models could be used as well.

can be reconsidered at product line evolution. Also, changing the product line architecture may lead to new architectural alternatives for the design of the products.

S7. Design Reference Architecture: The architectural decisions that are made to cover the whole range of variants implied in the variability decision model will be realized in the reference architecture.

4.3 Product Level

The major goal at the product level is to derive configurations based on the reference architecture to create particular products. At this level, the architecture of a concrete product may incorporate additional features apart from the base configuration. The following steps are leveraged to accomplish a product architecture:

S8. Resolve Variability Decisions: Let us assume that, at the product line level, the kind of variability decisions and architectural decisions described in the previous section have been determined. We can distinguish three situations for handling variability and architectural decisions: a) the variability identified at the product line level fits to the product level, thus we resolve the product line variability, b) the variability determined at the product line level has not yet supported all product-relevant functionality, however, the additional functionality is only relevant to a single product, or c) the variability identified in the previous step is insufficient and the needed variability is important for a range of products. The last case requires product line evolution [15]. Each situation will trigger the next steps for handling and resolving the variability and architectural decisions.

S9. Give Feedback, S10. Product Level Architectural Decisions: The case **S8(a)** is rather straightforward. In this case, fitting variability decisions have already been developed at the product line level. The decisions are taken and related to corresponding architectural decisions. Thus, selecting the variants constraints the architectural decisions through the mappings achieved in **S5**. If the variability decisions are sufficiently fine-grained, the architectural decisions can be automatically resolved. Otherwise, the architectural decisions are constrained and the architect performs a tradeoff decision among the remaining cases. In both cases the binding time of the architectural decisions can be the same or later than the variability decisions binding time.

S11. Further Product Level Architectural Decisions: The case **S8(b)** is related to additional product-specific functionality that needs to be designed. Therefore, the variability decisions do not provide further orientation as this is outside the scope of the functionality supported by reusable assets (hence *product-specific*). This case is not fundamentally different from architecting a single-system. The only distinguishing point is that the existing architectural decisions have to be considered. This can be resolved automatically as constraints among decision points and architectural options are available in the architectural decision model. The case **S8(c)** denotes that, at the product level, the capabilities provided by the reusable assets (and hence the variabilities) are insufficient. There are two possible approaches for handling this circumstance. In the ideal case, we can go back to the product line level and evolve the product line infrastructure to cover the special case. This would entail augmenting the variability model providing (if needed) additional architectural decisions and establishing the relations between them. As a result, the steps from **S3** to **S7** are repeated and the variability decision model, architectural decision model, and their interdependencies are reconsidered. Afterwards, the rest can be achieved similarly to the first situation. Nevertheless, sometimes, especially if there is an urgent need for shipping the product, a different decision can be made: changes are made at the product level that might raise inconsistencies at the product line level. In this case, the product line level should be evolved or adapted at a later point in time.

S12. Design Product Architecture: The resulting product architecture based on the reference architecture will be created based on both variability decisions and product-related architectural decisions made in the previous stages.

5. MOTIVATING EXAMPLE REVISITED

In this section, we present the implementation of the motivating example discussed in Section 3. For this, we have developed an integration of two tools, namely EASy-Producer—for variability management—and ADvISE—for architectural decision support. In the subsequent section, we describe the main features of these tools and demonstrate their integration.

5.1 EASy-Producer

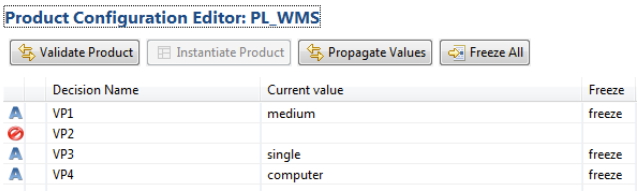
The EASy-Producer tool aims at providing modeling and realization support for software product lines and software ecosystems. It provides capabilities that are standard to all product line engineering tools, like variability modeling and support for the configuration process by determining consistency and consequences of a partial configuration (e.g., some value may be derived based on constraints and other given values). In addition, the tool has many capabilities that are well suited for our case. One is that it provides a very generic approach for instantiating artifacts (i.e., requirements, different forms of code, or, as in our case, architectural information). Another characteristic is that it has a powerful language for describing variability per se as well as constraints. The constraint language can also be used to describe implementation related decisions. While the tool can help to support consistency among the two levels, it allows for temporary violations. Thus, we can perform cases, where we first extend the product level and only later add it to the product line level.

5.2 Architectural Design Decision Support Framework (ADvISE)

ADvISE is an Eclipse-based tool that supports the modeling of reusable architectural decisions using Questions, Options and Criteria (QOC) [11] for systematizing the design space and providing decision support. In particular, it assists the architectural decision making process by introducing for a group of design issues a set of questions along with potential options, answers and related (often design pattern based) solutions, as well as dependencies and constraints between them. ADvISE has been developed with focus on reusable architectural knowledge that can be also transformed into reusable architecture designs rather than on product lines. However, it is generic enough to support both product line as well as product related architectural decision making. The advantage of the reusable architectural decision models is that the models need to be created only once for a recurring design situation. In similar application contexts, corresponding questionnaires can be automatically instantiated and used for making concrete decisions.

5.3 Decision Integration Tool Support

As discussed in Section 4, the first step in the tool support (cf. **S3**) is to represent the variability outlined in Table 1 (cf. **S1**) in the form of an EASy-Producer decision model. EASy-Producer supports two representations for variability models: an interactive view, where a configuration of the variability can be determined in an interactive manner (see Figure 4(a)) and a textual view where the variability can be described in a programmatic manner (see Figure 4(b)). In our example, four types of variability decisions



(a) Interactive view of the WMS example

```
1  project PL_WMS {
2
3      version v0;
4
5      enum PickingRateType {high, medium, low};
6      enum PartialPalletStrategyType {highSpeed, optimalReduction};
7      enum StaplerCraneStrategyType {single, multiple};
8      enum UIDeviceType {computer, mobile};
9
10     PickingRateType           VP1;
11     PartialPalletStrategyType VP2;
12     StaplerCraneStrategyType  VP3;
13     UIDeviceType              VP4;
14
15     enum BindingTime {designTime, compileTime, initTime, runTime};
16     attribute BindingTime bindingTime = BindingTime.designTime to PL_WMS;
17     assign (bindingTime = BindingTime.runTime) to {
18        PartialPalletStrategyType  VP2;
19     }
20 }
```

(b) Textual view of the WMS example

Figure 4: Variability model of the WMS example modeled in EASy-Producer

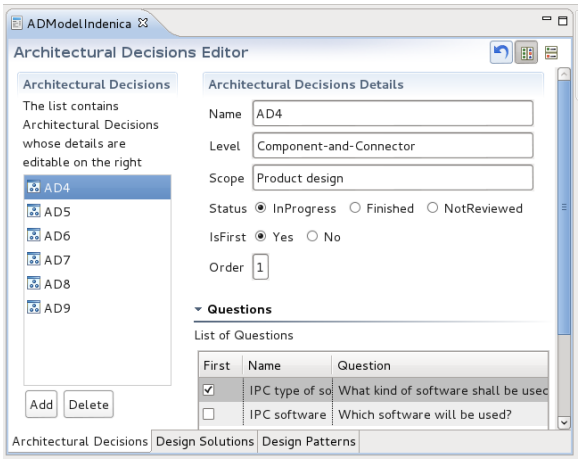
are defined ("PickingRateType", "PartialPalletStrategyType", "StaplerCraneStrategyType", and "UIDeviceType") along with their resolutions (lines 5–8). These types are used to define the variability decisions "VP1" to "VP4" of Table 1. The variability decisions "VP1", "VP3", and "VP4" will be bound at design time (lines 15–16) and "VP2" at runtime (lines 17–19).

Afterwards, we model the architectural decisions summarized in Table 2 and 3 (cf. **S2**) using ADvISE (cf. **S4**). The architectural decisions editor allows us to edit for each decision point a list of questions, and for each question a number of options which may be mapped to specific solutions and can be related to follow-on decisions and questions or constrained by other architectural options. In Figure 5, we give an example of the product-level architectural decision **AD4** of Table 3 that defines the types of IPC that can be used in the warehouse product. While Figure 5(a) gives general information about the underlying architectural decision, we can see the alternative options related to a specific question in the detailed view of Figure 5(b). In this example, the alternative options "IPC open source", "IPC medium price", and "IPC very expensive" are provided for the question "IPC type of software".

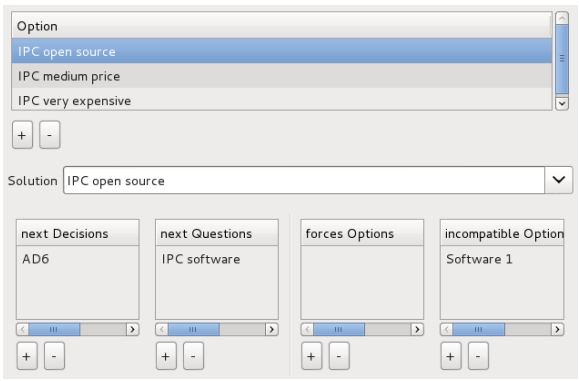
The next step of our approach (cf. **S5**) requires that we define the mapping between the variability and architectural decisions. For

```
<mappings>
  <aModel>ADModelIndenica</aModel>
  <vModel>PL_WMS</vModel>
  <vp name="VP1">
    <relation type="excludes">
      <vd id="PickingRateType.medium"/>
      <add id="AD4.IPC type of software.IPC
          open source"/>
    </relation>
  </vp>
  ...
</mappings>
```

Listing 1: Example of mapping between variability and architectural decisions



(a) Architectural decisions



(b) Options related to a question

Figure 5: ADvISE architectural decisions model editor

this purpose, we establish a set of mappings from the variability decision model elements (i.e., variants) onto the corresponding architectural decision model elements (i.e., architectural options) in XML format.

In particular, for each variability decision, relations of specific type (e.g., *excludes, enforces*, etc.) can be specified onto an architectural option. In Listing 1, the variability decision "PickingRateType.medium" is mapped to the architectural option "IPC open source" of the architectural decision **AD4** with type of relation "excludes". It means that selecting the *medium* picking rate will result in the rejection of *IPC open source*.

After designing the reference architecture to cover the whole range of variability (cf. **S6-S7**), it is now time to apply it to the problem of creating the product architecture. In step **S8**, we resolve the variability decisions by assigning values to the variability decisions. This is shown in Figure 6, where the lines 6–10 contain the decisions made for each variation point at design time. For instance, in our running example, the value "PickingRate.medium" was selected for "VP1". The case above actually corresponds only to step **S8(a)**, thus the configuration is straightforward. If we have the case of **S8(b)** or **S8(c)**, the situation becomes more complex. An example of **S8(b)** would be that our customer requires an integration to a specialized ERP system, while the product line typically only supports SAP-integration. Then, a typical approach would be to introduce an extension point and a connector to this specialized ERP-system. This would not necessarily be visible in the variability model or it would be simply modeled as an option to activate

the extension point. An example for **S8(c)** would be that a new customer would like to have a Partial Pallet Strategy, which is not yet supported, like max two (i.e., at most two pallets may be opened for one type of article). In order to support this in the future we would extend the "`PartialPalletStrategyType`" to include the "`maxTwo`" option.

The configuration given in Figure 6 defines the product from a variability perspective. Based on the connections to the architectural decisions a number of architectural decisions can be automatically derived and further ones can be constrained. In our example, the mapping we defined in Listing 1 will enable us to reflect variability decisions on the architectural decision model at the product level design (cf. **S9-S10**). To support architectural decision making, ADvISE tooling provides automatically generated questionnaires from the architectural decision models for guiding software architects. In Figure 7, the selection of the variant "`PickingRate-medium`" will cause the option "`IPC open source`" in the related ADvISE questionnaire to be deactivated. At this point, further architectural decisions for the product architectural design, related to variability or not, can be made (cf. **S11-S12**).

```
1  project WMS_product {
2
3      version v0;
4      import PL_WMS;
5
6      VP1 = PickingRateType.medium;
7      /* partialPalletStrategy = PartialPalletStrategyType.high;
8                 // This is not assigned, as this is only done at runtime */
9      VP3 = StaplerCraneStrategyType.single;
10     VP4 = UIDeviceType.computer;
11 }
```

Figure 6: Variability decisions in the WMS example

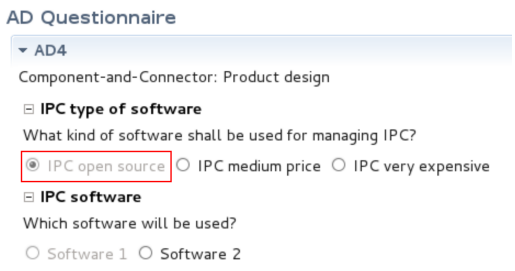


Figure 7: Architectural option deactivated due to variability decision

Discussion.

Through our motivating case, we observed that in many cases, interdependencies between variability and architectural decisions exist but are mainly kept implicit. Very often, these decisions are made by different stakeholders and with different tools and their overlaps and inconsistencies are resolved in an ad hoc and manual manner. We showed that formally eliciting the interdependencies requires additional efforts at the beginning but it leads to better automated support in integrating and harmonizing variability management and architectural decision making in the long run. By capturing and formalizing the links between variabilities and architectural options at the product line level, the architectural decision making process can be harmonized with the variability decision making process. As a result, architectural decisions can be changed or adapted whenever the variability is resolved. The advantage of our approach is that variability and architectural decisions remain consistent at product derivation. Also, the introduction of mappings between the two kinds of decisions can significantly enrich the documentation of the design rationale. For instance, the rejection or selection of an architectural option can be justified by following the dependencies with the corresponding variability decisions.

In our approach, we assume that the variability decisions guide the architectural decisions for the product line and product design. In practice, an architectural decision may also influence a variability decision. For instance, a decision to use a low-cost software solution because of cost constraints may cause some variabilities to be invalid. However, that would mean that the variability needs to be reconsidered and possibly redesigned (i.e., repeat steps **S3** to **S7** in Figure 3).

We discussed the interaction of variability and architectural decisions with the focus on product line design and product derivation and have not investigated the evolution and maintenance of product lines and products. As architectural decisions contain also interdependencies, reconsidering a variability decision may cause inconsistencies to existing architectural decisions. It is challenging to be able to handle this situation and also predict the impact variability decisions will have on the product architecture, but we plan to address this in our future work.

6. RELATED WORK

Variability and architectural decisions have been studied often in the literature in the same context. Variability decisions mainly refer to decisions related to the differences among the products that derive from a product line. The variabilities described as optional, alternative or multiple selections [9] are often related to architectural elements. For instance, the Feature-Oriented Domain Analysis (FODA) [8] usually mixes architecture-related decisions with domain properties and system configurations. Feature models mainly describe the solution space (i.e., focuses on modeling of commonalities and variabilities) and do not provide any guidance for selecting between alternative variants and reasoning about them. However, many approaches propose to enrich variability management with design rationale and decision support by introducing variability decision models [14]. In an approach that combines both methods, Perovich et al. [12] consider product line architectures as a set of architectural decisions, and use feature models to represent the decisions associated with the product features and transformation models to transform decisions into product architectures. The aforementioned approaches mainly focus on variability decisions and handle architectural decisions also as variability decisions without setting any boundary between the two.

Many approaches in the literature deal with modeling of reusable architectural decisions [21] or provide tool support for architectural decision making [17]. Unlike decision models for product lines that describe a set of variabilities relevant to product derivation, architectural decision models focus mainly on architecture-related options and alternatives for designing a software architecture. Some approaches suggest to integrate variability management with architectural knowledge and design rationale. For instance, Dhungana et al. suggest to capture variabilities in variability management as decisions and establish relationships between assets, such as components and decisions, explicitly [6]. Also, Capilla and Babar suggest to integrate architectural decision models with variability models to support ADDs for product line architectures [3]. For this, they map design decisions to variabilities and binding times to document reasoning about decisions related to product lines. A number of approaches in software product line engineering focus on documenting architectural design decisions and their rationale [3]. Unlike variability management approaches based on feature models [8] or decision models [14], these approaches propose to view architectural design decisions in modeling and managing of product

line variability models as first class citizens. For this, they distinguish between variations considered at product configuration and architectural decisions made at early stages of the design phase. As before, these approaches consider the design of product line and product architectures as an architectural decision making process and do not distinguish it from variability management. None of these approaches studies, elicits, and resolves the interdependence between variability and architectural decisions.

Some researchers have proposed the characterization of variability decisions according to the stage they are resolved. For example, Rosenmüller et al. apply multi-dimensional separation of concerns in variability modeling, that is, they create different variability models for different stakeholder concerns and use generalization and specialization mechanisms to model extension, composition and configuration of the variability dimensions [13]. Bidian et al. introduce variability decision boundaries at the different stages of requirements definition, architecture, and detailed design and runtime, according to their resolution time [2]. In both approaches, variability and architectural decisions are considered to have overlaps and interrelationships. However, our approach is the first proposal for investigating and supporting the interaction between these two kinds of decisions systematically.

7. CONCLUSIONS

In this paper, we studied the interdependence of variability and architectural decisions during product line and product design. Although variability decisions constraint and influence architectural decisions in practice, this inherent interdependence has not been studied or addressed systematically in the literature yet. We propose to make the interdependence of variability management and architectural decision making explicit and to manage variability and architectural decisions in an integrated manner. To ensure that variability and architectural decisions remain consistent to each other at product level, we introduce at the product line level dependency mappings between them, for instance, a specific variability decision may exclude or enforce a related architectural decision. These mappings are leveraged at the product derivation and give feedback—mainly introduce constraints—to the architectural decisions. In the context of a motivating scenario, we documented variability and architectural decisions and their interdependencies and demonstrated our approach using EASy-Producer, ADvISE, and their integration. In the future, we plan to study further these interdependencies in various case study scenarios, classify and formalize them. The systematic integration of architectural and variability decisions is important not only when creating reference architectures and derive products but also during the evolution and maintenance of product lines and products. We consider the latter to be an open challenge and plan to investigate different forms of integrating both kinds of decisions as well as the impact of changing these decisions during product line and product evolution.

Acknowledgement This work was partially supported by the EU FP7 project INDENICA (`http://www.indenica.eu`), grant no. 257483.

8. REFERENCES

[1] BACHMANN, F., AND BASS, L. Managing Variability in Software Architectures. In *Symposium on Software Reusability: Putting Software Reuse in Context (SSR)* (2001), ACM, pp. 126–132.

[2] BIDIAN, C., AND YU, E. S. K. Towards Variability Design as Decision Boundary Placement. In *10th Workshop on Requirements Engineering (WER)* (2007), pp. 139–148.

[3] CAPILLA, R., AND ALI BABAR, M. On the Role of Architectural Design Decisions in Software Product Line Engineering. In *2nd European Conference on Software Architecture (ECSA)* (2008), Springer, pp. 241–255.

[4] CLEMENTS, P., AND NORTHROP, L. *Software Product Lines: Practices and Patterns*. Addison-Wesley, Boston, MA, 2002.

[5] CZARNECKI, K., HELSEN, S., AND EISENECKER, U. Staged Configuration Through Specialization and Multi-Level Configuration of Feature Models. *Software Process Improvement and Practice 10*, 2 (2005), 143–169.

[6] DHUNGANA, D., GRÜNBACHER, P., AND RABISER, R. DecisionKing: A Flexible and Extensible Tool for Integrated Variability Modeling. In *1st International Workshop on Variability Modelling of Software-intensive Systems (VaMoS)* (2007), pp. 119–128.

[7] EL-SHARKAWY, S., DEDERICHS, S., AND SCHMID, K. From Feature Models to Decision Models and Back Again: An Analysis Based on Formal Transformations. In *16th International Software Product Line Conference (SPLC)* (2012), ACM, pp. 126–135.

[8] KANG, K. C., COHEN, S. G., HESS, J. A., NOVAK, W. E., AND PETERSON, A. S. Feature-Oriented Domain Analysis (FODA) Feasibility Study. Tech. rep., Carnegie-Mellon University Software Engineering Institute, November 1990.

[9] LINDEN, F., SCHMID, K., AND ROMMES, E. *Software Product Lines in Action - The Best Industrial Practice in Product Line Engineering*. Springer, 2007.

[10] LYTRA, I., SOBERNIG, S., AND ZDUN, U. Architectural Decision Making for Service-Based Platform Integration: A Qualitative Multi-Method Study. In *Joint 10th Working IEEE/IFIP Conference on Software Architecture & 6th European Conference on Software Architecture (WICSA/ECSA), Helsinki, Finland* (2012), IEEE.

[11] MACLEAN, A., YOUNG, R., BELLOTTI, V., AND MORAN, T. Questions, Options, and Criteria: Elements of Design Space Analysis. *Human-Computer Interaction 6* (1991), 201–250.

[12] PEROVICH, D., ROSSEL, P. O., AND BASTARRICA, M. C. Feature Model to Product Architectures: Applying MDE to Software Product Lines. In *Joint Working IEEE/IFIP Conference on Software Architecture European Conference on Software Architecture* (2009), vol. 11, IEEE, pp. 201–210.

[13] ROSENMÜLLER, M., SIEGMUND, N., THÜM, T., AND SAAKE, G. Multi-dimensional Variability Modeling. In *5th Workshop on Variability Modeling of Software-Intensive Systems (VaMoS)* (2011), ACM, pp. 11–20.

[14] SCHMID, K. A Comprehensive Product Line Scoping Approach and its Validation. In *International Conference on Software Engineering (ICSE'24)* (2002), ACM, pp. 593–603.

[15] SCHMID, K., AND EICHELBERGER, H. A Requirements-Based Taxonomy of Software Product Line Evolution. *Electronic Communications of the EASST 8* (2007).

[16] SCHMID, K., AND JOHN, I. A Customizable Approach to Full Lifecycle Variability Management. *Sci. Comput. Program. 53*, 3 (Dec. 2004), 259–284.

[17] SHAHIN, M., LIANG, P., AND KHAYYAMBASHI, M. R. Architectural Design Decision: Existing Models and Tools. In *Joint Working IEEE/IFIP Conference on Software Architecture and European Conference on Software Architecture (WICSA/ECSA), Cambridge, UK* (2009), IEEE, pp. 293–296.

[18] SINNEMA, M., VAN DER VEN, J., AND DEELSTRA, S. Using Variability Modeling Principles to Capture Architectural Knowledge. *Quality 31*, 5 (2006), 5.

[19] VAN GURP, J., BOSCH, J., AND SVAHNBERG, M. On the Notion of Variability in Software Product Lines. In *Working IEEE/IFIP Conference on Software Architecture (WICSA)* (2001), IEEE, pp. 45–54.

[20] ZDUN, U. Systematic Pattern Selection Using Pattern Language Grammars and Design Space Analysis. *Software Practice & Experience 37* (July 2007), 983–1016.

[21] ZIMMERMANN, O., GSCHWIND, T., KÜSTER, J., LEYMANN, F., AND SCHUSTER, N. Reusable Architectural Decision Models for Enterprise Application Development. In *3rd International Conference on Quality of Software Architectures (QoSA), Medford, MA, USA* (2007), Springer, pp. 15–32.