On the Role of Architectural Design Decisions in Software Product Line Engineering

Rafael Capilla[1] and Muhammad Ali Babar[2]

[1] Universidad Rey Juan Carlos, Spain
rafael.capilla@urjc.es
[2] LERO, UL, Ireland
malibaba@lero.ie

Abstract. An increased attention to documenting architectural design decisions and their rationale has resulted in several approaches and prototype tools for capturing and managing architectural knowledge. However, most of them are focused on architecting single products and little attention has been paid to include design decisions in the context of product line architectures. This paper reports our work on analyzing the existing work on architectural design decisions for the specific needs of software product line engineering. We have studied two existing tools for managing design decisions to identify the changes required in these tools for supporting product line specific requirements. Based on this study, we report the extensions required in the data models of the tools and propose a unified data model to guide the tool development research for supporting explicitly the relationships between design decisions and variability models for software product line engineering.

1 Introduction

Early in the nineties, Perry and Wolf emphasized the importance of Design Rationale (DR) in Software Architecture (SA) [23]. To date, the traditional approaches to documenting software architectures have been mainly based on the description of architecture views that reflect the interest of different stakeholders [11] [19] [24], but little attention has been paid to capturing and managing the rationale for key design decisions. The need to reduce the maintenance effort and to avoid architecture erosion because decisions are never recorded requires them to be captured.

Many claims have been made about the problems caused when design decisions are not explicitly documented [28], as they constitute a clear way to mitigate the effort required in understanding the architecture of a system when the experts or the creators of the architecture are no longer available. Bosch pinpoints [7] that "*we do not view a software architecture as a set of components and connectors, but rather as the composition of a set of architectural design decisions*". This idea, also stated in [11], claims for methods and techniques to enable the representation and capture of architectural design decisions in parallel with their architectures. In order to bridge the traditional gap between requirements and designs, a "new" architecture view so-called the "decision view", is proposed in [12], which considers design decisions as a cross-cutting information with respect to the other traditional architecture views that have to be documented explicitly.

Other research works rely on the definition of specific templates for capturing and representing the knowledge that should be part of the description of a design decision. Some research efforts (such as reported in [20] [31]) have proposed extensive list of attributes for characterizing the design decisions while others [9] advocate the use of more flexible approaches based on a list of mandatory and optional attributes that can be tailored for different organizations and user's needs in order to reduce the effort spent during the knowledge capturing activity. Kruchten et al. [20] consider that Architectural Knowledge (AK) = Design Decisions + Design, and they use ontologies to describe both the decisions and the relationships between them. Since design decisions can bridge the gap between requirements and architectures and code, recording these relationships [32] can benefit maintenance and evolution processes, and help to understand the root causes of changes or to estimate change impact analysis. To date, only a few researchers have partially addressed using design decisions in the context of software product line engineering. We assert that more work is required to providing methodological and tooling support for capturing and managing architectural design decisions in the context of product line engineering. In this paper, we propose a model for characterizing architectural design decisions in software product lines and compare two existing tools for possible extension to explicitly link architectural design decisions with product line variability models.

The rest of the paper is structured as follows. Section 2 identifies product line specific features of design decisions and some approaches that explicitly consider design decisions in product line practices. Section 3 discusses our approach to relate architectural design decisions to product line features. Section 4 discusses how existing tools for managing design decisions can support the specific needs of product lines. Section 5 describes a unified data model to guide the research on tooling support for design decisions for product line architectures. Section 6 discusses the related work and Section 7 describes our future direction for this line of research.

2 Product Line Engineering Features

Software Product Line Engineering (SPLE) has emerged as a successful mode of developing software. SPLE aims to create a family of related products based on a single architecture that can be tailored to meet the requirements of different products. This is achieved through the identification of commonalities and variations among the different systems of a given family which are represented in a product line architecture (PLA). The required variability can be realized using different variability realization techniques such as reported in [27]. A PLA has several unique features that allow the creation of multiple products by means of derivation techniques [6]. Here we discuss some of the key characteristics of PLA practices. Our discussion is focused only on those issues of SPLE that are closely related to the architecture development practices.

- **Variability modelling:** Product lines rely on the description of a set of common and variable characteristics of systems, and variability modelling deals with the representation of the common and variable aspects through a set of interrelated variation points and variants. Variability modelling is a challenging activity that needs suitable tooling support for managing the hundreds of variation points in complex systems. A major weaknesses of one of the most widely used modelling languages like UML, is that all variability concerns are difficult to be represented in a UML model. For instance, the relationships and constraints between variation points and variants that can be represented in a FODA tree [18] are hardly to be described in a UML diagram. Only the OCL (Object Constraint Language) combined with UML provides better support. A variability model is a decision model in which variation points and their variants have to be selected and instantiated for configuring a particular product.

- **Binding time:** This mechanism is used to delay design decisions as late as possible. The realization of variability can also be achieved through different binding times, such as; compilation, integration, deployment, or runtime. Different techniques can be used to resolve the binding time [15] of a particular software product. In general, binding times are not described in UML architecture models and they are documented separately from architectural designs.
- **Variability dependencies and product constraints:** Variation points and variants may depend on other variation points. Therefore, the selection of a certain variation point or variant may depend on a previous selection. These dependencies are usually represented in the variability model [17] [22]. Dependency models have a great impact on traceability as they provide viable paths to traverse from feature models to products and vice-versa. Frequently, dependencies are used to delimit the scope and the number of products during product configuration and mostly driven by economic and business factors. For instance, more complex rules like *if-then-else* conditions can be defined using a logic formula to specify product constraints. Such rules often crosscut variability models. Because product lines are market-driven, the domain scoping activity aims to define the number and types of products to be delivered with specific constraints. Hence, when making design decisions to produce a particular product, all the constraints should be resolved to avoid the selection of incompatible variation points that may lead to incompatible products.
- **Common and specific requirements:** Traditional software development use different types of requirements to motivate the design decisions made in the design phase. In product lines, only a subset of these requirements are common, whilst product specific requirements motivate the decisions to characterize each product by means of variation points defined in the modelling phase.

2.1 Reasoning Models in Product Lines Approaches

The majority of the approaches that try to capture design decisions alongside architectures do not consider the specificities of product lines. Only some proposals attempt to do this and they try to introducing reasoning in variability models. For instance, in [5] the authors describe a tool for supporting the analysis of feature models using an automatic reasoning mechanism to deal with extra functional features, like for instance quality aspects. This reasoning mechanism can be used to ask questions, such as: which is the number of potential products in a given feature model? Dhungana et al. [13] state the difficulty to transfer general architectural knowledge from tacit to an explicit form understandable by the users, and map this concept to product line variability models in which features have to be linked to architectural artifacts. More specifically, Dhungana et al. [14] perceive decisions as variation points for asset composition. These decisions are organized hierarchically and they become relevant if they are made in a certain order. The consequences of a decisions are expressed a logical dependencies. Decisions are represented in a decision model which only describes the need, the scope, and the constraints, but the same as in [5] no additional information about the rationale or the impact of the decision is supported. In [30], the authors address how to represent and document design decisions in product lines that follow a compositional approach to derive the final products. This composition is supported by the AHEAD tool suite [4], where product line features are used as building blocks of systems. In [30], decisions are documented as XML artifacts during the synthesis of the architecture. In their approach, text descriptions for understanding the decisions are included with product line assets. Finally, none of the tools analyzed in [10] for modeling and managing product line variability models considers the description of architectural design decisions as first class entities. In most cases, the tools only focus on how to deal with feature models or in product derivation tasks.

3 Design Decisions for Product Line Architectures

Because almost all the approaches described in section 2 suffer from the lack of support from explicit description of the decisions and their underlying reasons that accomplish the selection of variation points and variants in a product line, this section presents our attempt to link both concepts as a way to include the rationale used in architectural decisions with the definition and selection of variation points in the PLA. Hence, we propose to associate the concept of design decision with variability models in order to enrich the reasoning activity that takes place during product line modeling and derivation phases. We have also designed a general model that can be instantiated for building tool to support the proposed approach to incorporating the design decisions into variability models. Following are the elements of the proposed model.

- **Design decisions and variability modeling:** A variability model constitutes a decision model in which variation points and variants have to be resolved in order to achieve a specific product configuration. From our view, the definition of these variations should be considered fine grained decisions (as opposed to coarse grained decisions based on design patterns and architectural styles). Different levels of granularity in the decisions can be considered, but from an architecture point of view, the main decisions are precisely made in the early stages of the design phase for the core architecture whereas variation points are introduced later to refine the classes of a product line architecture. In our approach we propose a mapping between the design decisions that affect the definition or the selection of a variation point and variant in order to explain the reasons by which we arrived at a particular feature model.
- **Binding time:** The binding time should be understood in the traditional use of product lines, but the selection of a particular binding time should also be explained as a design decision. Hence, we believe that the binding time of a particular variation point should have a decision associated with it that explains

the realization of such variation point in the architecture. This binding time should not be confused with the time in which the decision is made, as all of them are defined at design time. We assume that the binding time of a set of related variation point should happen at the same moment. Otherwise, the realization of the variability will not be feasible.

- **Product constraints:** In the same way as we connect different decisions with specific relationships, we need to specify the links between variation points. Such links are usually defined by means of logical operators such as AND, OR, XOR, and NONE. More complex formulas can also be defined to support crosscutting relationships in the feature model or to include functional dependencies between features that usually happen during the execution of a system. These constraints delimit the scope of the products and may affect to previous decisions. From our viewpoint, a new design decision should document each product constraint to explain the rationale and the impact for such relationship in the variability model. Additionally, because one constraint may depend on a previous constraint in the variability model, the dependencies between design decisions could be also used to define the links between different product constraints and to avoid duplicate networks of dependencies.
- **Common and specific requirements:** In order to discriminate between requirements for a PLA and for a single product, we introduce a new attribute, to discriminate between common and specific requirements.

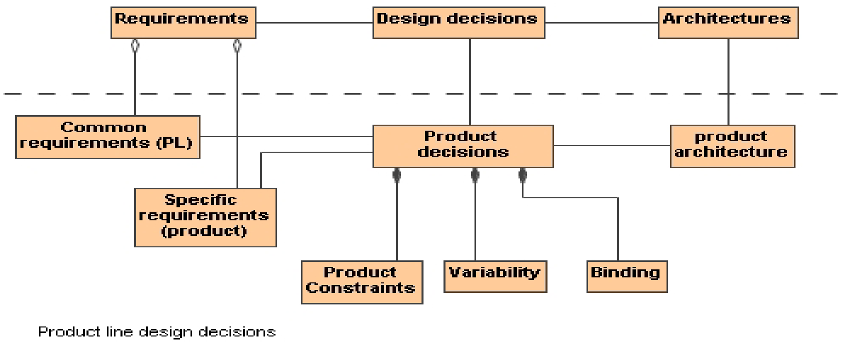

Fig. 1. A general model that maps the decisions to requirements and architectures and its corresponding relationships for product line architectures

The upper part of Figure 1 shows that design decisions are used to bridge the gap between requirements and architectures as requirements motivate the decisions that produce a particular architecture description. The lower part of the figure adds the additional entities that belong to SPLE. Common and specific product requirements motivate the decisions that lead to a concrete product architecture. These decisions are also design decisions but enhanced with some distinct features like decisions associates to: *product constraints, variability models, and binding time*. Similar traces between both parts of the figure provide a complete traceability among the entities. Our improvement in this general model is to map significant decisions to the product line variability model which need to be documented explicitly. We assert that this general model can be instantiated for building specific tool support.

4 Tool Support for Capturing Architectural Design Decisions

Having reviewed the existing variability management tools, we assert that none of them provide sufficient support for capturing and managing design decisions and the rationale that lead to the selection of any particular product line architecture or to a concrete variability model. Additionally, the research tools developed for managing architectural knowledge (such as Archium [16], AREL [29], ADDSS [8], and PAKME [3]) do not support product line features; neither do these tool support variability management. However, we believe that existing architectural knowledge management tools can easily be extended to support architectural decisions and their rationale in SPLE. In order to determine the requirements for extending the tools, we decided to analyze the data models of two architectural knowledge management tools (i.e., ADDSS and PAKME) we have developed. This Section provides brief descriptions of both tools and the level of support provided by them for SPLE.

4.1 Product-Line Support in ADDSS

ADDSS (Architecture Design Decision Support System [8] is a web-based tool for managing architectural design decisions (http://triana.escet.urjc.es/ADDSS). The tool supports an iterative process in which design decisions are captured along with their rationale and models. ADDSS supports basic dependencies between decisions and traceability between requirements, design decisions, and architectures. The chain of dependencies between decisions is documented explicitly for traceability purposes.

We have analyzed the current implementation of ADDSS to determine how it can support the SPLE features described in Section 3. Our conclusion is that in order to support the inclusion of variation points, variants and their relationships, the ADDSS data model can be easily extended to store such information and relate this with logical operators (such as AND, OR, XOR) to support product constraints and relationships between the variation points. Additionally, it is possible in ADDSS to associate each design decision to its respective variation points and variants, as a way to motivate and explain the definition or the selection of a particular variation point. However, relating parts of the variability model to a subset of the architecture is not possible because the current version of ADDSS cannot relate a set of design decisions to individual architectural parts. A complementary issue that needs to be addressed is to check the inconsistencies in the variability model to avoid incompatible product configurations. This should be implemented separately in order to ensure the integrity of the decision model for detecting unnecessary violations in the decisions when these are added, changed, or removed, but at present this feature is not supported yet. Also, including the binding time as an entity or attribute into the ADDSS data model should not be a problem, and the same stands for discriminating between common and specific requirements that are selected during the reasoning activity in the architecting process. We can also improve the documentation generated by the tool if we include the information belonging to the variability model of all product architectures.

4.2 Product-Line Support in PAKME

Process-centric Architecture Knowledge Management Environment, PAKME, is a web-based tool to support software architecture design, documentation and evaluation activities [3]. PAKME provides a knowledge repository, templates and features to capture, manage, and present architectural knowledge and design rationale. PAKME's knowledge repository is logically divided into two types of knowledge:

- **Generic:** such as general scenarios, quality attributes, design options; and
- **Project specific:** such as concrete scenarios, contextualized patterns, quality factors, architecture design decisions and rationale underpinning them.

Project-specific AK consists of the artefacts either instantiated from the generic knowledge or newly created during the software architecture process. Access to a repository of generic AK enables designers to use accumulated "wisdom" from different projects when devising or evaluating architecture decisions for projects in the same or similar domains. The project specific part of the repository captures and consolidates other AK artefacts and rationale such as concrete scenarios, design history, and findings of architecture evaluation. A project specific AK repository is also populated with knowledge drawn from an organisational repository, standard work products of the design process, logs of the deliberations and histories of documentation to build organisation's architecture design memory.

Though, PAKME has initially been developed to support the architecting activities for a single product, many of its artifacts (such as general scenarios, design options, and analysis models) can be used to support the activities for designing and evaluating PLAs. However, there needs to be certain changes required in the data model and interface for establishing and maintaining explicit relationships between different artifacts of a product specific architecture and PLAs. PAKME's data model also needs to be modified to accommodate the requirements of the solution proposed in Figure 1. Such changes can easily be accommodated as PAKME has successfully tailored to a specific domain and the experience showed such modifications are easy [1].

5 Design Rationale Support for Product Line Architectures

In order to assess the data models of PAKME and ADDSS for potential extension, we have used an illustrated example from the Intrada Product Family [25]. The objective of this assessment was to determine how variability, product constraints, and common and specific product requirements can be supported by both tools. We have identified the entities in both data models that support similar or completely different features in their respective tools. We have identified those entities that do not exist in either of the studied tools but required to support the architectural decisions in SPLE. Table 1 (i.e., Appendix 1) shows the entities associated concepts that are similar or equal in the studied tools as well as those which are not supported by either of the tool. The blue rows in Table 1 show the entities of PAKME not supported by ADDSS and light brown row identifies one feature of ADDSS not supported by PAKME).

5.1 A Unified Data Model for Architectural Design Decisions

Based on our analysis of the data models of ADDSS and PAKME, we have identified the minimum number of entities required to support architectural knowledge management by ADDSS and PAKME and left out other complementary entities that characterize concepts like quality attributes, quality factors, and findings included in PAKME's data model to support the software architecture evaluation activities. Table 2 (in Appendix 2) show the entities that comprise the proposed core data model to integrate the two tools for supporting architectural design decisions in SPLE. Figure 2 depicts the core data model using the UML class diagram notation.

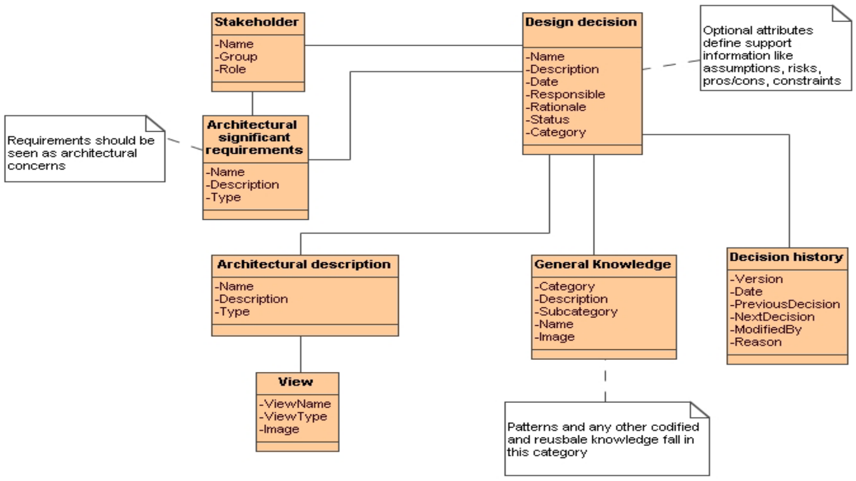

Fig. 2. Unified data model comprising the core entities that support architectural knowledge

Figure 2 shows that design decisions are related to architectures, requirements, and to the stakeholders that make the decision. Architecture descriptions are represented in terms of views, while design decisions comprise reusable chunks of knowledge like patterns and style. It should be noted that "rationale" of a decision is defined as an attribute in the proposed model, but from a higher level perspective, rationale is considered an entity in the ANSI/IEEE 1471-2000, currently under review. The data model support decision evolution by a specific entity, which records all the modifications and changes made to a particular decision. We have kept the data model simple and attributes of its entities as less as possible. However, the data model can easily be extended to support new features required for SPLE. In the next Section, we demonstrate how to extend the core model in Figure 2 to support the features of software product lines.

5.2 Extended Data Model to Support Product Line Features

We have extended the core data model characterizing architecture knowledge shown in Figure 2 to incorporate the features of SPLE mentioned in Section 3. We have refined the description given in Figure 1 to add the entities and attributes perceived necessary to link architectural design decisions to the product line variability model. The result of this customization is presented in Figure 3, which shows that we have added three different entities as well as some attributes to cater the new needs. One new entity captures the information representing the variations points, including a constraint rule that defines the logical relationship between the variants. In addition, a *category* attribute is used to perform a classification of different variation points that can be filtered for visualization purposes. This *variation point* decision is attached to the architectural design decision which explains and motivates the definition of a particular variation point.

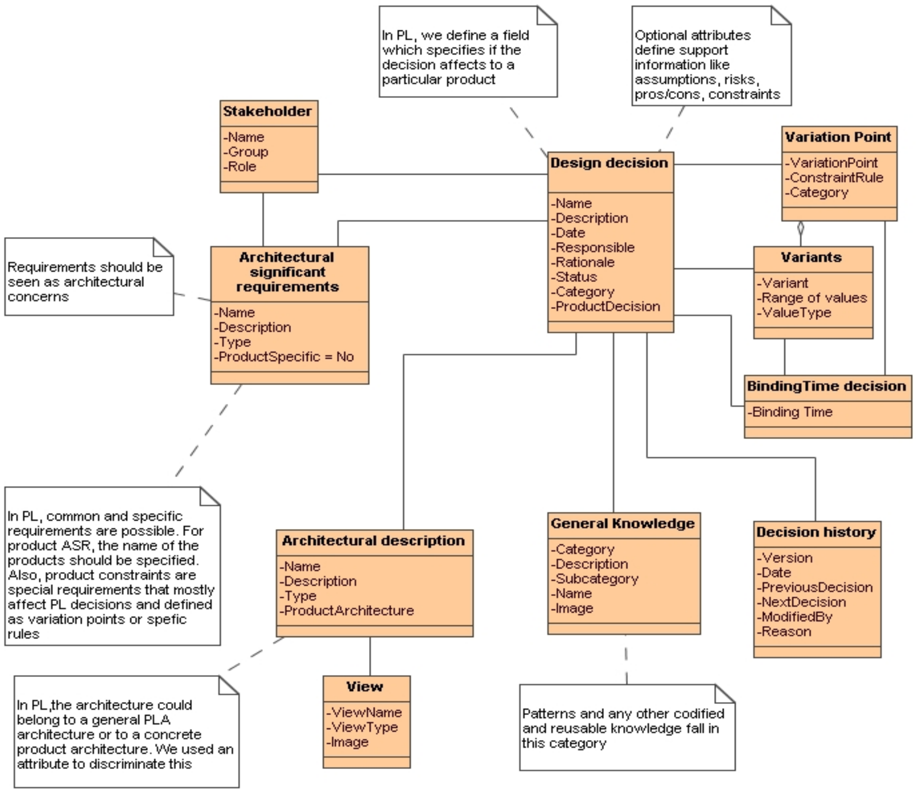
In PL, we define a field which specifies if the decision affects to a particular product
Optional attributes define support information like assumptions, risks, pros/cons, constraints
Stakeholder
-Name
-Group
-Role
Design decision
-Name
-Description
-Date
-Responsible
-Rationale
-Status
-Category
-ProductDecision
Variation Point
-VariationPoint
-ConstraintRule
-Category
Variants
-Variant
-Range of values
-ValueType
Requirements should be seen as architectural concerns
Architectural significant requirements
-Name
-Description
-Type
-ProductSpecific = No
BindingTime decision
-Binding Time
In PL, common and specific requirements are possible. For product ASR, the name of the products should be specified. Also, product constraints are special requirements that mostly affect PL decisions and defined as variation points or spefic rules
Architectural description
-Name
-Description
-Type
-ProductArchitecture
General Knowledge
-Category
-Description
-Subcategory
-Name
-Image
Decision history
-Version
-Date
-PreviousDecision
-NextDecision
-ModifiedBy
-Reason
In PL,the architecture could belong to a general PLA architecture or to a concrete product architecture. We used an attribute to discriminate this
View
-ViewName
-ViewType
-Image
Patterns and any other codified and reusable knowledge fall in this category

Fig. 3. Extended data model to incorporate product line feature in design decisions model

We took similar steps for incorporating the variants defined in the variability model. This fine-grained decision has also its corresponding architectural design decision which justifies the selection of particular variant in the architecture. A new *binding time decision* class was added to indicate the binding time of a particular variation point or variant, which is related to the architectural design decision that indicates why a binding time has been chosen, but also to the variant or variation point affected by such binding time. Therefore, we can achieve a fine-granularity to define different binding times for different variants and variation points. The *design decision* class has also a new attribute, *ProductDecision*, which defines if a design decision concerns to a product line decisions ("yes") or to a single architecture ("no"). Other two entities provide attributes for supporting additional product line features. The *architectural significant requirement* class uses an attribute ("ProductSpecific") to distinguish between common requirements for the entire line or for a single product, or a product specific requirement. Finally, the *architectural description* entity uses a new attribute to indicate if the description belongs to a PLA (this will imply that the variability model has been resolved and the decisions are made). The data model shown in Figure 3 does not consider multi-product lines (a product line composed of product lines). Neither does it represent the dependencies between design decisions as these are supported by PAKME and ADDSS as intermediate classes that relate one decision to others decisions. The same also applies to defining relationships between different variation points. These aspects have been left out to avoid cluttering.

5.3 Impact of Product Line Features in the Reasoning Activity

This Section presents our analysis of the impact that product line features may have on the reasoning process. In addition to the information aimed to support product line decisions, the process by which a variation point or variant is selected or configured would have some influence on the steps of the reasoning activity. For instance, typical architectural design decisions are selected after an evaluation of the best or optimal choices among several design options. However, the instantiation of a variation point only depends of the selection of their variants and values. Hence, no alternative design decisions are evaluated or stored for this case. Otherwise, the selection of a particular binding time may imply to consider and evaluate different binding time alternatives, which might not be the same for different architectures of subsystems. Additionally, the existing architectural knowledge management tools like PAKME and ADDSS must capture new relationships that are now established between the variability model and the decisions that explain such variability model. To make the decision capturing process more agile, these relationships should be defined internally by the tools to alleviate the effort spent by a user to define new links. Only in those cases where a relationship between variation points and variants has to be defined, a user must reason about such dependency and make it explicit when capturing the decision. Such effort can also be reduced if the dependency that models a relationship in the variability model also serves to model the dependency between the decisions attached to variation points and variants. Hence, product lines slightly modified the way in which dependencies are modeled with respect to single architecture, as the architect would define the relationship in the variability model and the tool implicitly uses that relationship to internally establish the relationships with its associated decisions.

6 Related Work

Recently, many researchers have emphasized the importance of treating architectural design decisions as first class entity, however, only a few have mentioned the use of design decisions in SPLE. In [2], the authors present a framework for representing design decisions in a SPL, which is structured as a design decision tree (DDT) where nodes represent design decisions and branches relate nodes to each other. Some of these nodes represent the variations in a product family. Lago and Vliet [21] show how to introduce assumptions in UML models representing architectures and variability concerns in SPLE. They present assumptions with feature models to show the influence of the assumptions on the features used to model variability. The work reported in [30] focuses on the reuse of design decisions in order to customize product line using composition techniques as a step-wise refinement for product derivation. Design decisions are captured in XML files that can be reused during the transformations needed to obtain a final product. Extensions to products and their design decisions can easily be traced by viewing the ways decisions extend architectures through successive refinements. The COVAMOF model for managing variability described in [26] is used to support the notion of architectural design decisions in a SPL. The authors map architectural concepts (including decisions) to COVAMOF concepts to demonstrate the feasibility of capturing architectural knowledge and link this to variability models by mapping similar concepts. All these approaches attempt to map conceptual entities related to architectural design decisions to product line variability models and even introduce some basic notation to explain the selection of the variation points, but all of them lack detailed guidance on how to model and explicitly represent the relationships between design decisions and a product line variability model. Though, some of these approaches use existing PL infrastructure to add information about design rationale, they do not provide explicit data model and conceptual guidance about extending tools to relate the product line features with decision models. These are some of the shortcoming of the existing approaches to describing architectural decisions and variability modeling in SPLE that has been addressed by the work reported in this paper. We believe that the presented data model can provide sufficient guidance to extend the current tools or develop new ones to support explicit relationships between architectural design decisions and variability models along with rationale for those decisions.

7 Conclusions

This paper presents the continuation of our efforts in integrating two similar tools for capturing and documenting architectural design decisions. Because there is a lack of specific support for product line architecture decisions, we have merged the data models of both tools in order to distill a common data model. The proposed unified model supports the decisions made in a product line context in order to explain the decisions made in variability models. Thus, thriving research area provides the necessary infrastructure for systematically and rigorously incorporating the notion of design decisions and their rationale in designing and maintaining PLAs. Our work identifies the characteristics of architecture design decisions in the context of SPLE. From the common model based on the data models of two architectural knowledge management tools, we have observed that it is not very difficult to incorporate the decisions made in a variability model to support the specificities of product lines. However, accommodating this new information may result in a large number of medium-size or fine grained decisions. Based on our research on the role of architectural design decisions in SPLE and providing appropriate tool support, we believe that the same dependencies defined for the architectural design decisions can be used to define the relationships in the variability model in order avoid introducing duplicate dependency links. Additionally, supporting common and specific requirements is also necessary to distinguish those decisions that are specific to a single product. For future work in this line of research, we intend to build a web-based tool by integrating ADDSS and PAKME's features and extending them in order to support new ones based on the proposed unified data model and test the new capabilities in a product line environment.

Acknowledgements

Ali Babar's work is supported by Science Foundation Ireland under grant number 03/CE2/I303-1. The work of Rafael Capilla is partially funded by the PILOH project of the Spanish Ministry of Education and Research programme under grant number URJC-CM-2006-CET-0603.

References

1. Ali-Babar, M., Northway, A., Gorton, I., Heurer, P., Nguyen, T.: Introducing Tool Support for Managing Architectural Knowledge: An Experience Report. In: Proceedings of the 15th IEEE International Conference on Engineering Computer-Based Systems, Belfast, Northern Ireland (2008)
2. Alonso, A., León, G., Dueñas, J.C.: Framework for Documenting Design Decisions in product Families Development. In: ICECSS, pp. 206–211. IEEE CS, Los Alamitos (1997)
3. Babar, M.A., Gorton, I.: A Tool for Managing Software Architecture Knowledge. In: Proceedings of the 2nd Workshop on Sharing and Reusing Architectural Knowledge, ICSE Workshops (2007)
4. Batory, D.S., Sarvela, J.N., Rauschmayer, A.: Scaling Step-Wise Refinement. IEEE Transanctions on Software Engineering 30(6), 355–371 (2004)
5. Benavides, D., Trinidad, P., Ruiz Cortés, A.: Automated Reasoning on Feature Models. In: Pastor, Ó., Falcão e Cunha, J. (eds.) CAiSE 2005. LNCS, vol. 3520, pp. 491–503. Springer, Heidelberg (2005)
6. Bosch, J.: Design and Use of Software Architectures. Addison-Wesley, Reading (2000)
7. Bosch, J.: Software Architecture: The Next Step. In: Oquendo, F., Warboys, B.C., Morrison, R. (eds.) EWSA 2004. LNCS, vol. 3047, pp. 194–199. Springer, Heidelberg (2004)
8. Capilla, R., Nava, F., Pérez, S., Dueñas, J.C.: A Web-based Tool for Managing Architectural Design Decisions. In: Proceedings of the Workshop on Sharing and Reusing Architectural Knowledge. ACM Digital Library, Software Engineering Notes, vol. 31(5)
9. Capilla, R., Nava, F., Dueñas, J.C.: Modeling and Documenting the Evolution of Architectural Design Decisions. In: Proceedings of the 2nd Workshop on Sharing and Reusing Architectural Knowledge, ICSE Workshops (2007)
10. Capilla, R., Sánchez, A., Dueñas, J.C.: An Analysis of Variability Modelling and Management Tools for Product Line Development. In: Proceedings of the Software and Services Variability Management Workshop – Concept Models and Tools. Helsinki University of Technology Software Business and Engineering Institut, Helsinki, Finland, HUT-SoberIT-A3, pp. 32–47 (2007) ISBN: 978-951-22-8747-5

11. Clements, P., Bachman, F., Bass, L., Garlan, D., Ivers, J., Little, R., Nord, R., Stafford, J.: Documenting Software Architectures. Views and Beyond. Addison-Wesley, Reading (2003)
12. Dueñas, J.C., Capilla, R.: The Decision View of Software Architecture. In: Oquendo, F., Warboys, B.C., Morrison, R. (eds.) EWSA 2004. LNCS, vol. 3047, pp. 222–230. Springer, Heidelberg (2004)
13. Dhungana, R.R.D., Grünbacher, P., Prähofer, H., Federspiel, C., Lehner, K.: Architectural Knowledge in Product Line Engineering: An Industrial Case Study. In: Euromicro Conference on Software Engineering and Advanced Applications, pp. 186–197 (2006)
14. Dhungana, D., Grünbacher, P., Rabiser, R.: DecisionKing: A Flexible and Extensible Tool for Integrated Variability Model. In: Proceedings of the 1st Workshop on Variability Modelling of Software-intensive Systems (VAMOS), LERO, UL, Ireland (2007)
15. Fritsch, C., Lehn, A., Strohm, T.: Evaluating Variability Implementation Mechanisms. In: Procs. of International Workshop on Product Line Engineering (PLEES 2002), Technical Report at Fraunhofer IESE (No. 056.02/E), pp. 59-64 (2002)
16. Jansen, A., Bosch, J.: Software Architecture as a Set of Architectural Design Decisions. In: 5th IEEE/IFIP Working Conference on Software Architecture, pp. 109–118 (2005)
17. Jaring, M., Bosch, J.: Variability Dependencies in Product Family Engineering. In: van der Linden, F.J. (ed.) PFE 2003. LNCS, vol. 3014, pp. 81–97. Springer, Heidelberg (2004)
18. Kang, K.C., Cohen, S., Hess, J.A., Novak, W.E., Peterson, A.S.: Featured-Oriented Domain Analysis (FODA) Feasibility Study. Technical Report, CMU/SEI-90-TR-21 ESD-90-TR-22, Software Engineering Institute, Carnegie Mellon University, Pittsburgh (1990)
19. Kruchten, P.: Architectural Blueprints. The "4+1" View Model of Software Architecture. IEEE Software 12(6), 42–50 (1995)
20. Kruchten, P., Lago, P., van Vliet, H.: Building up and Reasoning About Architectural Knowledge. In: Hofmeister, C., Crnković, I., Reussner, R. (eds.) QoSA 2006. LNCS, vol. 4214, pp. 43–58. Springer, Heidelberg (2006)
21. Lago, P., van Vliet, H.: Explicit Assumptions Enrich Architectural Models. In: Inverardi, P., Jazayeri, M. (eds.) ICSE 2005. LNCS, vol. 4309, pp. 206–214. Springer, Heidelberg (2006)
22. Lee, K., Kang, K.C.: Feature Dependency Analysis for Product Line Component Design. In: Bosch, J., Krueger, C. (eds.) ICOIN 2004 and ICSR 2004. LNCS, vol. 3107, pp. 69–85. Springer, Heidelberg (2004)
23. Perry, D.E., Wolf, A.L.: Foundations for the Study of Software Architecture. Software Engineering Notes, ACM SIGSOFT, pp. 40–52 (October 1992)
24. Rozanski, N., Woods, E.: Software Systems Architecture: Working with Stakeholders Using viewpoints and Perspectives. Addison-Wesley, Reading (2005)
25. Sinemma, M., Deelstra, S., Nijhuis, J., Bosch, J.: COVAMOF: A Framework for Modeling Variability in Software Product Families. In: Nord, R.L. (ed.) SPLC 2004. LNCS, vol. 3154, pp. 197–213. Springer, Heidelberg (2004)
26. Sinemma, M., van der Ven, J.S., Deelstra, S.: Using Variability Modeling Principles to Capture Architectural Knowledge. In: 1st SHARK Workshop (2006)
27. Svahnberg, M., van Gurp, J., Bosch, J.: A Taxonomy of Variability Realization Techniques. Software Practice & Experience 35(8), 705–754 (2005)
28. Tang, A., Babar, M.A., Gorton, I., Han, J.A.: A Survey of the Use and Documentation of Architecture Design Rationale. In: 5th IEEE/IFIP Working Conference on Software Architecture (2005)
29. Tang, A., Jin, Y., Han, J.: A rationale-based architecture model for design traceability and reasoning. Journal of Systems and Software 80(6), 918–934
30. Trujillo, S., Azanza, M., Diaz, O., Capilla, R.: Exploring Extensibility of Architectural Design Decisions. In: Proceedings of the Workshop on Sharing and Reusing Architectural Knowledge and Design Intent (SHARK/ADI 2007), ICSE Workshops, Minneapolis, USA, May 2007. IEEE CS, Los Alamitos (2007)

31. Tyree, J., Akerman, A.: Architecture Decisions: Demystifying Architecture. IEEE Software 22(2), 19–27 (2005)
32. Wang, A., Sherdil, K., Madhavji, N.H.: ACCA: An Architecture-centric Concern Analysis Method. In: 5th IEEE/IFIP Working Conference on Software Architecture (2005)

Appendix 1: PAKME-ADDSS Data-Model Comparison

Table 1. Comparison of the entities supported in both PAKME and ADDSS data models

PAKME	PAKME description	ADDSS	ADDSS description	Matched
Stakeholder	People interested in the architecture process or product	Users	Stakeholders with different roles interested in architectures	Yes
Stakeholder Group	Define the project access rights	Permissions	Rights for each user type	Yes
Architectural significant requirements (ASR)	Are NFR (QA) Quality goals can be derived. An ASR must be satisfied by one or several design decisions	Requirements	Functional and non-functional requirements. Only the type, the number, and a text description is provided	Yes
Scenario	Is a refinement of ASR and are QA	NFR	A non-functional requirement	Partially through requirements
Quality factors	Are the factors a QA should match			Only quality attributes are supported
Findings	A description for the QA that meet a particular scenario			Not supported in ADDSS
Analysis Model	Is a reasoning framework that reasons about the effect of different tactics on QA scenarios			Not supported in ADDSS
Design Tactic	Is a design mechanism for achieving the desired level			Not supported in ADDSS
Pattern	Characterizes a design solution in a given context	Pattern	Describes a design solution. Patterns are classified by its type, description and an usage example	Yes
Effect of pattern	Defines the effect of the pattern on a particular QA			Not supported in ADDSS
Support Information	Captures the background information required to justify the choice of a decision for a particular scenario	Optional attributes	Optional attributes capture extra information	Partially supported
Architecture Decision	Is a high level decision that satisfies FR and NFR. There are	Design Decision	Captures the design decision and its rationale through a set	Yes

Table 1. (*continued*)

	dependencies between decisions.		of attributes. Basic dependencies can be defined but not the of the dependency	
Architecture Decision Rationale	Is the reason behind the architecture	Rationale	Is the reason behind the architecture	Yes
Design history	A history of decisions is supported	Version and responsible	Some attributes in ADDSS are used for the same goal	Partially supported
Alternative	Design decisions may be related to other design alternatives	Status and category attributes	ADDSS offers a category attribute to indicate if a decision is alternative design choice but also an status to know if the decisions has been approved or rejected	Yes
Architecture Description	Prescribes the architecture to be realized	Architecture	Provides the information about a particular architecture and a link to the views supported	Yes
Architectural View	Provides a description for architectural views with the images	View attribute	Describes the view of the architecture and provides a link to it	Yes
		Translation	Provides multilingual support	Not supported in PAKME

Appendix 2: Core Common Entities between PAKME and ADDSS

Table 2. Main common entities distilled from PAKME and ADDSS data models

Common entity	Entity description
Stakeholder	Are those persons interested in the architecture process or product
Architectural significant requirements	Functional and non functional architectural significant requirements drive the selected design decisions
General Knowledge	Characterizes a design solution in a given context, like patterns or styles
Design Decision	Is a high level design decision that explains the decisions and its underpinning rationale
Decision history	A history of decisions is supported
Architectural Description	Prescribes the architecture to be realized
View	Provides a description for architectural views with the images