Modeling Product-Line Architectural Knowledge

Jessica Díaz, Jennifer Pérez, Juan Garbajosa, Carlos Fernández-Sánchez
Technical University of Madrid (UPM), CITSEM, ETSISI, Madrid, Spain
yesica.diaz@upm.es, jenifer-perez, jgs}@eui.upm.es, carlos.fernandez@upm.es

Abstract

The documentation of architectural knowledge helps to understand and prevent the violation of previous decisions. Documenting the architectural knowledge of product lines is more complex than in the case of single-product architectures because it requires considering the existing variability in a product family. This implies having the capability to describe design decisions and relationships related to variability. This paper presents the concept of Product-Line Architectural Knowledge (PLAK): the knowledge of product line architecture and its variability. This concept has been realized through a solution based on models and traceability between models called the PLAK Model.

1. Introduction

In contrast to traditional definitions of software architecture [3], new definitions agree that software architecture should be seen “as the result of a set of design decisions rather than a set of components and connectors” [4]. As a result, most researchers in the area emphasize the need for documenting architectural knowledge [1]—i.e., design decisions, rationale, assumptions, constraints, etc.—to improve the understanding and rationalization of architectural decisions made during the design of an architecture solution. This could help architects to reason about change, thus maintaining and evolving the architecture while preserving its integrity [1][22].

Despite the fact that there has been a clear progress in architecture documentation [23], the documentation of architectures used to realize software product lines (SPL) [20] is still a challenge. Product line architecture (PLA) must serve the needs of many potential products of the same family or domain; therefore, it must describe the commonality of these products but also their differences to make explicit the variation points among the products. Therefore, documenting the knowledge of PLA requires documenting the design decisions and rationale behind the commonality—i.e., the rationale behind making specific decisions to design the existing variations in the product line. Documenting single-product architectures has a high strategic value and deep impact on the success of software product lines. However, common approaches to documenting architectural knowledge do not specifically support variability [1][23]. A few approaches partially address the documentation of variability design rationale [5][14] as well as the traceability of the life of variations [16] from the requirements (i.e., the problem space) to the architecture (i.e., the solution space). The work by Capilla and Babar [5] and Lytra et al. [14] supports defining relations between variability and architectural decisions. However, the specification of variability is not embedded in the architecture models, and thus it is not possible to trace variability decisions to architectural elements. Finally, Perovich et al. [19] encode design rationale as model transformations from features to architecture, and thus they only document the decisions and the traceability of those variations that specially emerge from the problem space (characteristics or qualities visible to customers).

To completely support the documentation and traceability of variability design rationale, it is necessary to consider the variability that emerges from the problem space and also from the solution space—i.e., variability that is only visible to the engineers [20], specifically to the architects, such as that related to tactics, patterns, technologies, and tradeoffs between quality attributes, among others. To make explicit this architectural variability, mechanisms to embed variability in the architecture specification model are necessary. To that end, this paper presents a new concept: Product-Line Architectural Knowledge (PLAK), i.e., the knowledge of a PLA and its variability. This knowledge is explicitly documented through a solution based on models and traceability between models, called the PLAK Model, which is supported by the FPLA (Flexible Product-Line Architecture) modeling framework. The PLAK Model defines a set of types of design decisions that support the documentation of variability design rationale and act, when necessary, as traceability links between the requirements and architecture.

1 Available on https://syst.eui.upm.es/FPLA/home
Documenting PLAK helps understand and preserve the integrity of the architecture during its construction and evolution, whereas traceability facilitates the selection and resolution of variants (aka. binding) to automatically configure and generate the customized product applications of a product line [7]. We put the PLAK Model into practice in a software factory in an industrial project for developing a product line for autonomic power grids. Validation is performed through a case study that shows how the PLAK Model helped preserve the integrity of architecture during the construction and evolution of this product line.

The paper is structured as follows: Section 2 presents a state-of-the-art on managing variability, Section 3 describes the PLAK Model, Section 4 contents the case study, and Section 5 concludes.

2. State of the art

Managing variability requires specifying variability for different stakeholder concerns, from the problem space (requirements) to the solution space (architecture), as well as documenting its rationale and traceability [20]. A set of approaches—e.g., the Orthogonal Variability Model [20] and the Variation Model [2]—claim that variability should be independently specified in a separate model. However, although these approaches effectively work to document the variability of the problem space and trace this variability with the other development artifacts, they present limitations when it is necessary to specify variations that emerge, for instance, from the design decisions of an architect. By contrast, variability as an integral part of development artifacts supports different views of variability for different stakeholders’ concerns, providing specific mechanisms for each kind or artifact and reducing complexity by hiding or showing fine-grained variations. Hence, feature models are generally used to describe requirements in terms of common and variable customer-visible features of a product line [11]. Figure 1.a shows a feature model of a product line for home automation systems. These features are refined and realized in a PLA—defined as a set of reusable components and connectors [3] that can be configured to build the products that make up a product line. The realization of each feature typically requires several finer-grained architectural decisions. Figure 1.b shows a PLA for home automation systems where variability is embedded in the architecture specification model, specifically in the component and connector view of software architectures.

How variability is realized in PLA largely determines how variability can be documented and traced. There has been extensive research on supporting variability in PLA, as the review of the state of the art that Schaefer et al. [10] presents. Most of these mechanisms specify variations in the architecture by modifying its configuration via selecting optional, alternative, or multiple components and their
respective connectors (see the optional component \texttt{ApplianceControl} in Figure 1.b). We refer to these kinds of variations as \textit{architectural external variability}. However, this external variability is not enough to completely define all kinds of variations. This happens when variations have a lower granularity than components so that they are materialized inside simple components—or non-composite components. As a result, it is necessary to specify variations that are internal to components. We refer to this kind of variation as \textit{architectural internal variability}. In these components where variability occurs inside, part of their functionality is common to the product line and part of their functionality changes depending on the product to be derived. In this regard, a previous work called the Flexible-PLA Model \textbf{Error! Reference source not found.}. allows one to capture external and internal variability as part of PLA. The Flexible-PLA Model is based on the principles of \textit{aspect-oriented software development} \cite{12}. The main concept underlying the Flexible-PLA Model is a \textit{plastic partial component} (PPC), which is a solution to completely support the internal variation of architectural components. The variability of a PPC is specified using \textit{variation points} that hook fragments of code known as \textit{variants} to the PPC through \textit{weavings} that specify where and when to extend the PPCs using the variants. As a result, variants can crosscut and be reused by several PPCs of the PLA. These concepts are exemplified in Figure 1.b, which shows the graphical representation of two PPCs, \texttt{DoorLock} and \texttt{Surveillance}, that define two variation points with two variants, \texttt{KeyPad} and \texttt{FingerprintScanner}, which can be used both for blocking the doors and a security box.

Previous work on documenting and tracing variability design rationale \cite{14,16,19} does not consider variability as a first citizen of the architecture, together with its documentation and traceability. This makes it difficult to document and trace the design rationale of the variability that emerges from architecture—only visible to the engineers—both externally and internally. In fact, if \textit{architectural internal variability} is neither described, documented, nor traced, then important concerns, often related to quality attributes (e.g., portability, maintainability), could remain unknown and the knowledge surrounding it could evaporate.

\section{The PLAK Model}

The PLAK Model is a solution for the definition, management, and traceability of the knowledge of a PLA and its variability. It reuses common elements of design decisions that previous work defined \cite{1} such as rationale, assumptions, constraints, and alternatives; further, it defines three new types of design decisions that capture variability design rationale and their relationships. These decisions are illustrated in Figure \ref{fig:plakmodel}, which shows the documentation of the \textit{home automation PLA} described in Figure 1.b. The three types of design decisions (DDs) are as follows: \textbf{Open DDs} document the design of the variation points of product lines. These decisions are left open to optional, alternative, or multiple variants. For example, Figure 2 illustrates the Open DD \texttt{Security} that keeps the rationale behind using the \textit{aspect-oriented paradigm} \cite{12} to implement the variability of the \texttt{IntrusionDetection} modes (e.g., basic and advanced) and the methods to implement them (e.g., door lock and surveillance). \textbf{Optional DDs} document the rationale of these variants. For example, Figure 2 illustrates two Optional DDs for implementing the variation of the \texttt{IntrusionDetection} modes. The DD \texttt{BasicMode} shows that the design of the basic mode is more economical but less secure, whereas the DD \texttt{AdvancedMode} is more secure and expensive because camera recording may require large data storage. \textbf{Closed DDs} document the design of the common structure for all the products of a product line. These decisions are closed to variation or further expected change.\footnote{This does not mean that Closed DDs cannot evolve to support unexpected changes.} For example, Figure 2 illustrates the Closed DD \textit{Event-driven architecture} that keeps the rationale behind implementing an event-driven architecture in order to asynchronously receive critical events from sensors (e.g., motion sensors). These three types of DDs offer complete support for documenting the commonality and variability of software product lines as follows: (i) These decisions are embedded in the same architecture description language used to specify PLAs—i.e., the Flexible-PLA Model (see Figure 2 and Section 2). This allows architects to document the variability rationale that emerges from architecture design—i.e., styles, patterns, technical solutions, and architectural quality attributes, among others—inside the architecture specification, and additionally, to document the knowledge of internal and external variability; and (ii) These design decisions act as traceability links from features to PLA \cite{7}, filling in the gap between the problem and the solution spaces. Figure 2 illustrates how the feature \texttt{Advanced} is traced through the Optional DD \texttt{AdvancedMode} with the variants \texttt{FingerprintScanner} and \texttt{Camera}.

Therefore, the PLAK Model allows one to consider variability as a first-class citizen of the architecture, together with its documentation and traceability; this provides the \textit{opportunity} to initiate the process for configuring customized products to derive individual product applications at the architectural design level.
Hence, PLAK supports the selection and resolution of variants for configuring product applications (aka. binding) through the design decisions instead of the components and connectors of the PLA. This approach performs the binding through the architects’ selection of the design decisions instead of selecting each of the variants of the PLA that they want to incorporate in a specific product one by one. This has the advantage of providing architects with greater power of abstraction, reducing the number of selections. Hence, architects just select the design decisions (Optional DDs); then, thanks to the traceability links, all the architectural variants associated with these design decisions are automatically selected (using model queries). Nevertheless, the binding time and how this binding is performed is not constrained to the design time, so it can be intentionally delayed thanks to the attributes binding time and binding technique that Open DDs provide. For example, Figure 2 shows the Open DD Security that specifies that the binding of the implementation mode for IntrusionDetection will be performed at compiling time through the weaving mechanism of aspect-oriented programming. As a result, Open DDs are bound to their variants, configuring specific product applications when the binding time is established—design time, compiling time, linking time, or running time—, and how the technique is specified—extensions, parameterization, configuration, etc., or aspect-oriented programming [12], as the Flexible-PLA Model proposes.

To be able to use these modeling primitives, we defined a domain-specific (modeling) language (DSL) through the specification of a metamodel (see Figure 3). The PLAK metamodel defines the metaclass DesignDecision that offers the primitives to instantiate a design decision, as well as the metaclasses OpenDesignDecision, ClosedDesignDecision, OptionalDesignDecision, and AlternativeDesignDecision, which inherit from DesignDecision (see Figure 3). The metaclass OpenDesignDecision defines two properties, BindingTime and BindingTechnique, and is composed of a set of Optional DDs. Any kind of design decision consists of alternatives. The PLAK Model also entails the definition of dependencies between two design decisions (see the relationship dependsOn). Additionally, these design decisions entail a set of elements that are needed to completely support reasoning and rationalization of the rationale behind them: Constraint, Assumption, Rationale (why, cost, risk, and tradeoff), Design, and Pattern.

Figure 2: The PLAK model for a home automation SPL.

Figure 3: PLAK metamodel.
The PLAK metamodel has been formalized based on the MOF architecture; therefore, any approach that is formalized through a metamodel following the MOF architecture could import PLAK concepts and use them by extending their metamodels or by model transformation. To enable its use, PLAK is supported by the FPLA modeling framework, which implements a graphical language for modeling knowledge (see Figure 2). FPLA is an open-source modeling tool that is available as an Eclipse plug-in. FPLA assists architects in describing PLA and documenting PLAK and guides them in defining the traceability links between feature and PLA models.

4. Case study

This section aims to provide empirical evidence about how helpful the PLAK Model is in order to improve the understanding of architectural design decisions and prevent the violation of previous design decisions during the design and evolution of the architecture. Toward that end, we used the case study technique [21]. Hence, a case study was conducted in an experimental software factory (iSSF [15]) in the Technical University of Madrid (UPM) and Indra Software Labs at the Spanish node. The iSSF is a global and distributed software development initiative set up at the end of 2011. The iSSF initiative aims to put models and tools into practice and track the results. Therefore, it is a suitable setting to deploy, track, and evaluate the feasibility of the PLAK Model. The case study was performed within an industrial project to develop a product line for autonomic power grids and lasted six months. The authors of this paper have been involved with this particular investigation since 2010. Next, the case study is reported according to the guidelines stipulated by Runeson and Höst [21].

4.1 Case Study Design

The research question to be answered through the case study analysis is formulated as follows: *Do PLAK models assist and guide architects (or the development team in general in the case of interdisciplinary teams) in understanding the design decisions and preventing the violation of previous design decisions during the design and evolution of the architecture?*

This research question aims to find out if the documentation of PLAK and being traced and embedded in the architecture was really useful for enabling the understanding the PLA during its construction while at the same time preventing the violation of previous design decisions. As a dependent variable, the *level of assistance and guidance* is qualitatively estimated by analyzing questions asked to the development team through a set of interviews. These questions asked about specific situations to analyze the assistance and guidance provided by the PLAK that architects had previously modeled. Hence, the team was asked if the PLAK models helped them when trying to maintain the architecture's integrity when changes were implemented. Specifically, the questions were focused on dependencies, assumptions, trade-offs, or constraints of those design decisions that may conflict with a proposed change and therefore could jeopardize the integrity of the PLA. The potential independent variables that might influence the dependent variable are the architects’ expertise, the project size, and the PLA complexity.

4.1.1 Data collection procedure. Both quantitative and qualitative data were gathered. The collection methods that were used are as follows: 1) Observation: two observers attended planning, architecting, and review meetings as well visited the development team twice weekly; 2) Interview: the team was interviewed following a questionnaire open to discussion; then, the interviews were transcribed and analyzed using the *constant comparison method*; 3) Archival data: the information about the project was collected in Redmine; 4) Analysis of work artifacts: the PLAK models generated with the FPLA modeling framework and the code under subversion were gathered.

4.1.2 Analysis & Validity procedure. Qualitative analysis was used to examine the data gathered. The procedure to explore the *chain of evidence* [21] from collected data is described as follows: interviews and meetings were recorded, transcribed, grouped by quotes, and coded. Coding means that parts of the text are given a code representing a certain topic of interest—one code is usually assigned to many pieces of text, one piece of text can be assigned more than one code, and codes can form a hierarchy of codes and sub-codes [21]. The coded material is enriched with comments and reflections (i.e., *memos*). Additionally, we used the *constant comparison method* to compare the data collected from different members of the team.

Additionally, as data gathered in case studies are mainly qualitative, and typically less precise than quantitative data, it is important to use *triangulation* to increase the precision of the study.

---

3. [https://syst.eui.upm.es/FPLA/download](https://syst.eui.upm.es/FPLA/download)

4. [http://www.upm.es/internacional](http://www.upm.es/internacional)


6. This paper only describes the most relevant facts from the report due to space limitations.
We used three types of triangulation: 1) Methodological triangulation, i.e., the use of different methods to measure the same concern, such as interviews, observations, and the analysis or archival data; 2) Data source triangulation, i.e., the use of multiple data sources at potentially different occasions by interviewing the team both separately and together; and 3) Observer triangulation, i.e., the use of more than one observer in the case study to replicate specific data collection sessions—including interviews—by two different observers.

4.1.3 Case study description. This case study is part of the large Spanish national project Technologies for the Automated and Smart Management of the Future Distributed Power Networks (ENERGOS)\(^\text{10}\) in which more than 30 industrial and academic partners were involved, including utilities, retailers, and software providers. ENERGOS aims to construct the basis of a framework to support the design of Smart Grids by providing reasoning capabilities to the software components that manage the operation of the different stakeholders involved in the grid (e.g., consumers, producers, storage, etc.).

Specifically, the case study focused on the first goal through the development of a product line for autonomic power grids during a six-month period. We approached the development of this product line through a bottom-up approach [13] that consists of starting with one or more products from which it is possible to generate the product line’s core assets and finally future products. The first product we developed with this approach was the platform that supports the autonomic behavior of a microgrid for the self-balancing of the microgrid when the power supply of the main network falls down.

Next, we briefly describe the microgrid and its software architecture. The microgrid is located in three towns in the south of Spain (for more details, see Error! Reference source not found.). This microgrid is divided into two feeder sections connected by an exchange point (see the deployment view of the architecture in Figure 4). Through this point, the two sections are connected to the main power network. The microgrid is composed of two substations, 40 transformation centers for 2,316 customers of low voltage, 53 transformation centers for 53 customers of medium voltage, three section centers of medium voltage, two photovoltaic generation plants, two storage systems, one prosumer (an industrial unit of microgeneration), and 93 demand managers (see Figure 4). The team designed the architecture for the autonomic behavior of this microgrid by incorporating autonomic managers [9] into the software components and connectors deployed in the microgrid (see Figure 4). Through these autonomic managers, distributed resources (i.e., energy producers, consumers, prosumers, storage, etc.) are autonomously orchestrated to maximize the time supplying energy by negotiating, coordinating, and balancing the energy that is produced, consumed, and stored.

4.1.4 Subject description. The development team was composed of ten people: seven of them more focused on development tasks and the other three more
focused on analysis and architectural design tasks. Additionally, two observers—a professor and a researcher from the Technical University of Madrid—and three experts on the domain of power grids from Indra Software Labs were involved. The expertise of the members of the development team varies between 0–1 years for the developers (most of them are graduate students engaged in vocational training) and 5–7 years for the analysts and architects. The most experienced members have participated in European projects on smart grids for the last few years.\textsuperscript{11,12}

4.2 Results

4.2.1 Case study execution. This section describes the use of the PLAK Model to document the Component & Connector view of the software architecture that implements an autonomic microgrid. The model resulting from documenting the knowledge and rationale generated during the architecting process through the different levels of decomposition of the system is shown in Figure 5. Level 1 shows some features, the main architectural elements that make up the software architecture of the microgrid under study, and the design decision traceability links among them. Focusing on architecture, Level 1 shows (i) the components GridController, Storage, Prosumer, Photovoltaic, MVConsumer, and LVConsumer; (ii) the plastic partial component (PPC) Substation; and (iii) the connectors ExchangePoint, DemandManager, Transformer, SectionCenter, and StorageManager.

Level 2 shows the design of the complex connector DemandManager, which is composed of the PPC AutonomicManager. Level 3 shows the design of the complex PPC AutonomicManager, which is composed of the PPCs Monitoring, Analysis, Planning, and Execution. Finally, Level 4 shows the design of the PPC monitoring, which is composed of the PPCs Listener, Filter, Translator, and Router.

In this case study, a total of 83 DDs were documented using the PLAK Model: 25 Closed DDs, 16 Open DDs, and 42 Optional DDs. Based on Figure 5, several of these decisions are described. The closed design decision A (see marker A in Figure 5) captures the rationale about why an event-driven MAPE-K Control Loop \cite{9} was selected to design the GridController, as well as some assumptions (e.g., the modes in which a microgrid can work in relation to the main power network—connected or island), tradeoffs (e.g., complexity), and constraints (e.g., real-time constraints). The closed design decision B (see marker B in Figure 5) captures the rationale about why a service-oriented MAPE-K Control Loop was selected to design the DemandManager, as well as risks and tradeoffs. During the design of this PLA—although starting from a unique microgrid—these design decisions were envisaged as common decisions to all future expected microgrids; thus, they are Closed DDs.

The design decision C is an Open DD (see marker C in Figure 5). It captures the rationale about why the aspect-oriented paradigm \cite{12} was selected to implement availability as a crosscutting concern. This decision responds to the future expected variability on requirements depending on the strictness of the quality feature availability (see marker D in Figure 5). This Open DD has two Optional DDs (see markers E and F) that maintain the rationale of why the team selected the architectural tactics active redundancy and passive redundancy to implement strict and non-strict availability (see CMU/SEI-2009-TR-006, 2009), as well as the alternative—the exception tactic—that was considered during the architecting process (see marker G). Open DDs can be bound to Optional DDs to derive a valid product configuration. It is necessary to highlight that it is possible to specify design decision dependencies—e.g., see the dependency between the design decisions B and C via an arrow between them. This means that the availability tactic to be implemented by the PPC substation depends on the communication interface that this component implements, i.e., the tactic will be different depending on whether a component implements a request-response pattern—service-oriented—or a publish-subscribe pattern—event-driven.

Finally, at Level 4, the open design decisions H–J (see markers H–J in Figure 5) are a clear example of variability that emerges from the architecture and not from features (requirements). This variability was specified for reuse of sub-components and connections that constitute the PPC Monitoring (see Level 4, Figure 5) of the PPC AutonomicManager (see Level 3, Figure 5), which supports the configuration of the monitoring properties for each specific autonomic manager. Hence, the team designed the PPC Listener with the variation points SensorType and CommMiddleware in order to be able to implement the autonomy of the grid controllers, which need to listen to fault detections (see marker H); storage managers, which need to listen to the storage level (see marker I); or photovoltaic plants, which need to listen to the production level (see marker J). These design decisions allowed the team to keep the traceability (also, the team maintained the rationale) of this architectural variability.

\textsuperscript{11} Intelligent Monitoring of Power NETworks http://innovationenergy.org/imponet/

\textsuperscript{12} NEtworked MOnitoring & COntrol, Diagnostic for Electrical Distribution. http://innovationenergy.org/nemocoded/
Figure 5: PLAK model of the microgrid under study.
4.2.2 Analysis and interpretation. This section shows the analysis of the interviews with the ten members of the development team. The interviews were recorded, transcribed, and coded to apply the constant comparison method. Each interview was individually coded by adding the codes and categories of codes to each paragraph of the interview by using the features of Atlas.ti. We obtained 28 codes, such as architectural knowledge, traceability, variability, useful, helpful, time-consuming, learning curve, and easy-to-understand (see codes and groundedness—the number of repetition of a code—in Figure 6). Then, the interviews were compared and the results combined.

From this analysis, we would like to emphasize those codes that allowed us to answer the research question (see Section 4.1). With regard to the code time-consuming, four interviewees said that the process of modeling features, design decisions, and architecture; tracing these artifacts; and linking architecture with code is time-consuming. With regard to the code easy-to-understand, six interviewees said that the design decisions that were documented by architects facilitated understanding of the development of the autonomic manager, and two interviewees said that these design decisions facilitated understanding of the development of the quality attribute availability. Finally, five interviewees emphasized the usefulness of the PLAK model and traces during the changes made in several architectural decisions as a consequence of the complexity and multitude of elements in the microgrid, as this excerpt illustrates: “the design decisions that we documented some months ago allowed us to understand the changes in the specification of the autonomic behavior upon detection of a failure in the main power network.” The analysis of these interviews provides evidence that the PLAK Model was helpful for the development and evolution of a product line for autonomic power grids. However, the team also highlighted the time consumed with the construction and evolution of the product line. Although time-consuming, the team positively valued the usefulness of having the PLAK model to revisit previous design decisions, as it was able to understand such a complicated system. However, the use of the PLAK Model requires knowing and understanding the modeling concepts on which it is based as well as learning the usage of the FPLA modeling framework. The learning curve of these concepts as well as the usage of FPLA could slow down the process of putting the PLAK Model into practice.

Figure 6: Snapshot from Atlas.ti (Codes).

4.3 Case study conclusions
The feasibility of the PLAK Model is proven through a case study run in a software factory. The case study puts the PLAK Model into practice for the development of a product line for autonomic power grids. The results prove that PLAK provided knowledge that helped the development team to understand the architectural design decisions during the construction and evolution of the product line. Although time-consuming, the team positively valued the usefulness of having the PLAK model to revisit previous design decisions, as it was able to understand such a complicated system. However, the use of the PLAK Model requires knowing and understanding the modeling concepts on which it is based as well as learning the usage of the FPLA modeling framework. The learning curve of these concepts as well as the usage of FPLA could slow down the process of putting the PLAK Model into practice.

5. Conclusion and Further work
This work dealt with the documentation and traceability of the architectural knowledge that is present in PLAs. Hence, the PLAK Model supports (i) the documentation of variability coming from the domain—i.e., features—and also variability coming from the design decisions and the architecture, as PLAK is embedded inside the architecture specification; and (ii) the traceability of the external variability of the architecture configuration as well as the internal variability of simple components, their

---

13 Atlas.ti is a tightly integrated suite of tools that support analysis of written texts, audio, video, and graphic data.
dependencies, and rationale. The documentation of internal variability is especially relevant, as it is often related to the variation of quality concerns that may crosscut the PLA. Both the documentation of design decisions associated with internal variations and the capability to trace the life of these variations had not been dealt with so far. As explained before, the PLAK Model has been formalized through MOF; therefore, it is possible to export the PLAK concepts to other architecture models.

This work also presents the evaluation of the feasibility of the PLAK Model through a case study in a software factory. Through this case study, we obtained some evidence about the guidance and assistance that the PLAK Model provided architects. The case study allowed us to conclude that the documentation of PLAK might improve the understanding of design decisions and provides the basis for reasoning about future design decisions or changes, including those design decisions that affect the variability of a product line, which helps preserve the violation of previous decisions. A trade-off between the effort necessary to document PLAK and the benefit resulting from such documentation must be analyzed in each project. As several studies show [8], this trade-off is dependent on the context of the projects and the stakeholders.

As PLAK is explicitly codified, reasoning can be automated or semi-automated, and as this knowledge is also traced, then change impact analysis can be effectively aided. In the future, we plan to extend the FPLA modeling framework via a pattern repository in order to facilitate the reuse of design decisions. Finally, we will also analyze the use of model transformations to specify design decisions by means of transformations.

6. Acknowledgments

ENERGOS (CEN-20091048), MESC DPI2013-47450-C2-2-R, and UPM Research Programme sponsored the work reported here.

7. References